# ELECTRONIC WARFARE ASSET ALLOCATION WITH HUMAN-SWARM INTERACTION

A Thesis

Submitted to the Faculty

of

Purdue University

by

William M. Boler

In Partial Fulfillment of the

Requirements for the Degree

of

Masters of Science Electrical and Computer Engineering

May 2018

Purdue University

Indianapolis, Indiana

# THE PURDUE UNIVERSITY GRADUATE SCHOOL
# STATEMENT OF COMMITTEE APPROVAL

Dr. Lauren A. Christopher, Chair

Department of Electrical and Computer Engineering

Dr. Brian S. King

Department of Electrical and Computer Engineering

Dr. Paul Salama

Department of Electrical and Computer Engineering

**Approved by:**

Dr. Brian S. King

Head of the Graduate Program

To my mother Janet, my father Michael, and my sister Cassandra. You have been my inspiration.

# ACKNOWLEDGMENTS

I would like to acknowledge my professor Dr. Christopher for providing me this opportunity to work with this project, and for her guidance and wisdom shared in making this project successful. I would like to thank Dr. King and Dr. Salama for their mentorship and tutelage. I would like to acknowledge Calvin Wieczorek and Md Saiful Islam for being a dedicated partners and contributors in this work; Jonah Crespo for his guidance and original contributions to *human in the swarm*; Joshua Reynolds for his original contributions to the initial GUI; Dr. Eberhart and Dr. Kennedy, the inventors of PSO; and Scot Hawkins for his guidance in EW propagations and his sponsorship. I also would like to acknowledge NSWC Crane for providing the support necessary for completing this project.

TABLE OF CONTENTS

Page

# LIST OF TABLES

# LIST OF FIGURES

Figure Page

Figure Page

# SYMBOLS

| | |
|---|---|
| $C_1$ | Constant for multiplying personal best component |
| $C_2$ | Constant for multiplying neighborhood best component |
| $p_i$ | Particle's best position |
| $p_{gb}$ | Particle global-best position |
| $p_n$ | Particle's neighborhood best position |
| $\mathbb{R}^3$ | The space of reals in X, Y, and Z |
| $\mathbb{R}^4$ | The space of reals in X, Y, Z, and Frequency |
| $\mathbb{R}^6$ | The space of reals in X, Y, Z, Frequency, Azimuth, and Elevation |
| $x_i$ | Particle position/state |
| $v_i$ | Particle velocity/state-derivative |

# ABBREVIATIONS

| | |
|---|---|
| 2D | Two-dimensional space |
| 3D | Three-dimensional space |
| DOD | Department of Defense |
| EW | Electronic Warfare |
| EWAAP | Electronic Warfare Asset Allocation Problem |
| GB | Global best solution |
| LB | Local best |
| Mhz | Megahertz |
| NB | Neighborhood Best |
| PSO | Particle Swarm Optimization |
| NP | Nondeterministic polynomial time |
| RX | Receiver |
| RF | Radio Frequency |
| TX | Transmitter |
| XCVR | Transceiver |

# GLOSSARY

| | |
|---|---|
| cost/fitness | An objective value for rating the "goodness" of particles against each other and used interchangeably in this text |
| local/neighborhood best | The personal best of a particle within the neighborhood of the current particle |
| local optima | A peak or valley in the solution space that does not represent the most optimum solution |
| global optima | A peak or valley in the solution space that represe–nts the most optimum solution |
| particle position | A particular state or solution from the problem space represented by the particle |
| particle velocity | The change of state from one particle position to the next |
| personal best | The personal best state found for the particle |

# ABSTRACT

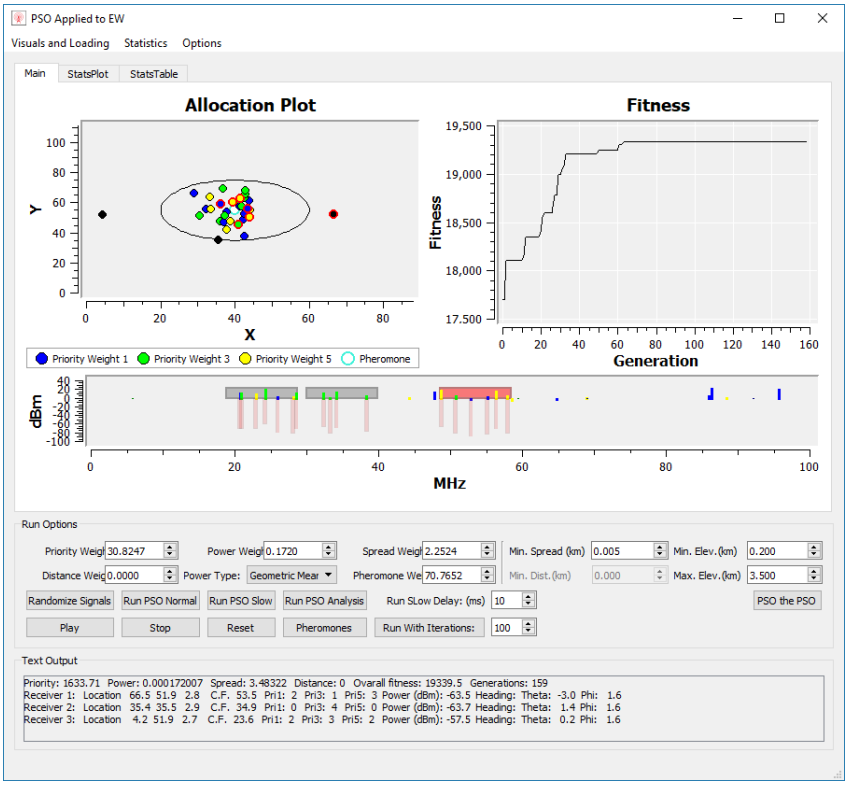
Boler, William M. M.S.E.C.E., Purdue University, May 2018. Electronic Warfare Asset Allocation with Human-Swarm Interaction. Major Professor: Lauren Christopher.

Finding the optimal placement of receiving assets among transmitting targets in a three-dimensional (3D) space is a complex and dynamic problem that is solved in this work. The placement of assets in $\mathbb{R}^6$ to optimize the best coverage of transmitting targets requires the placement in 3D-spatiality, center frequency assignment, and antenna azimuth and elevation orientation, with respect to power coverage at the receiver without overloading the feed-horn, maintaining sufficient power sensitivity levels, and maintaining terrain constraints. Further complexities result from the human-user having necessary and time-constrained knowledge to real-world conditions unknown to the problem space, such as enemy positions or special targets, resulting in the requirement of the user to interact with the solution convergence in some fashion. Particle Swarm Optimization (PSO) approaches this problem with accurate and rapid approximation to the electronic warfare asset allocation problem (EWAAP) with near-real-time solution convergence using a linear combination of weighted components for fitness comparison and particles representative of asset configurations. Finally, optimizing the weights for the fitness function requires the use of unsupervised machine learning techniques to reduce the complexity of assigning a fitness function using a Meta-PSO. The result of this work implements a more realistic asset allocation problem with directional antenna and complex terrain constraints that is able to converge on a solution on average in $488.7167 \pm 15.6580$ ms and has a standard deviation of 15.3901 for asset positions across solutions.

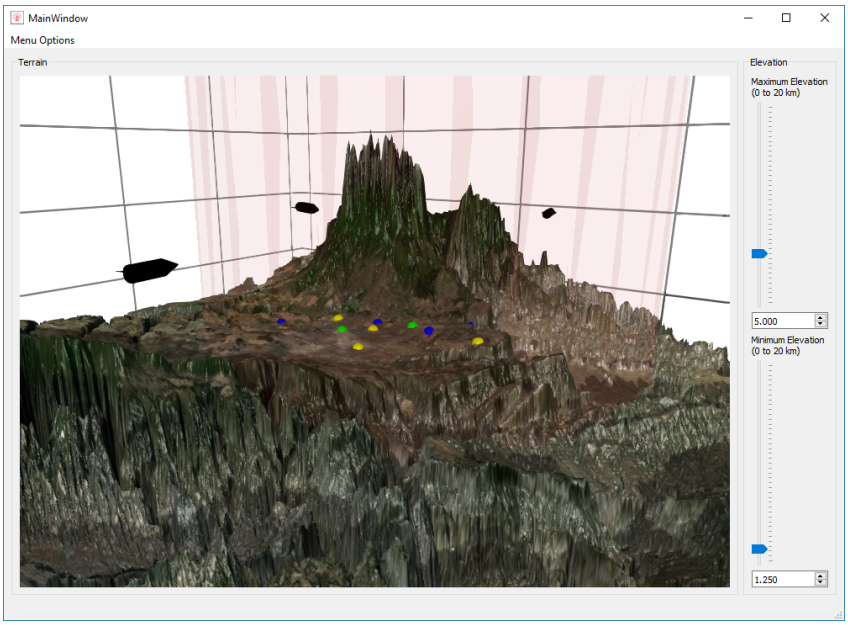
# 1. INTRODUCTION

## 1.1 Overview and Problem Statement

The asset allocation problem in electronic warfare (EW) involves the assignment of transmitters to receivers in a $\mathbb{R}^6$ environment, with respect to $xyz$-spatiality, frequency assignment, and heading. Receiving or jamming assets must be placed in optimal locations to maintain adequate coverage of transmitter targets with respect to spatial position and bandwidth assignment. Optimization of the asset placements is an NP-Complete problem that is reducible to the traveling salesman problem (TSP). Comparable to TSP, each receiver-transmitter assignment must be analyzed with respect to power received, receiver sensitivity, target priority, receiver feed-horn power limitations, spatial positioning, frequency and bandwidth coverage, and terrain constraints to find the best solution. A brute-force approach to analyzing assignments can quickly become intractable. As such, an approximation to the real-world solution is approached. Particle Swarm Optimization (PSO) is considered for finding an accurate approximation to the real-world best solution. An application of PSO for use in approximating signal assignments for receiving assets among transmitting targets in an EW environment is presented, with modifications made to original work improving the realism and presentation of solutions while maintaining under one second performance metrics.




(a) Main Window




(b) 3D Display

Fig. 1.1. (a) provides the main window for program control, fitness across generations, and solution display. (b) provides the 3D display of the solution and terrain.

Figure 1.1 provides the current state of the program. Figure 1.1(a) shows the 2D solution representation, the fitness across each generation of the solution, the spectrum coverage of each asset in the solution, inputs for fitness function weights, the minimum spread distance, the minimum and maximum elevations, buttons for running PSO, and a text output of the solutions. In the "Allocation Plot", yellow, green, and blue represent prioritized target transmitters, black circles represent the assets in the solution, and red outlines indicate signal assignments. Figure 1.1(b) shows the 3D solution with a pheromone keep-away zone shown as a red, transparent cylinder. Pheromones and keep-away zones are to be explained in Chapter 3.

## 1.2 Project Inheritance and Continuation

This work is a continuation of research provided by [1, 2]. Reynolds [1] provides the preliminary 2D environment for the optimization of receivers among transmitters using Qt. Crespo [2] made advancements with implementing preliminary human-interaction and 3D terrain constraints. The inherited work provided a visual display of 2D placements of assets, signals, keep-away boundaries, a fitness plot of the prescribed solution, buttons for executing tasks, an options dialog box for updating parameters, and an output text-box for displaying solution results and other messages.

Continuation work provided by this team modified the existing program to display 3D environments using Qt Data Visualization, implement advanced terrain models collected from ArcGIS, evaluate and implement multi-threading techniques, and provide frequency-hopping characteristics while maintaining real-time performance metrics. Further work resulted in [3, 4]: Christopher et al. [3] implemented the initial implementations of pheromone interactions; Witcher [4] implemented matching of real-time simulated assets to solutions using the Hungarian Algorithm. Work in terrain models, display, and multi-threading are explained by Wieczorek [5].

This thesis describes the implementations of human interaction with solution convergence via *pheromones*, meta-PSO for fitness function optimization, eradication of

the distance fitness component, adjustments to the power calculations and power fitness component, statistics-collecting libraries, and initial implementations of directed antenna. Furthermore, portions of original work are re-factored to optimize the code for increased readability and modularity. The additions described in this thesis maintain the under one-second benchmark requirement of solution convergence, while increasing the readability and reuse within the code, resulting in an overall reduced complexity of adding future modifications and increased realism of solutions.

## 1.3 Literature Review

### 1.3.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization is an evolutionary computational technique for finding approximations to complex problem spaces [6,7]. It was developed by Dr. Russell Eberhart and Dr. James Kennedy in 1995 in an effort to model the flocking behavior of birds and the swarming behavior of insects [6]. PSO has been used in a variety of applications [1, 2, 8–17]. PSO provides fast convergence to optimal solutions and works in a relatively simple paradigm for rapidly solving accurate approximations to complex and dynamic real-world problems.

The basic principle of PSO works as follows. A population of particles, each representative of a particular solution to the problem, is initialized and flown in the hyper-dimensional problem space. Each particle is evaluated by an objective fitness[1] function, which provides a measure of the "goodness" of the particle with respect to other particles. The objective fitness function is a model of the real-world, but holds the property that it does not need to exactly calculate the real-world value[2]. Each particle remembers its personal best position and its neighbor's best position. After evaluation, each particle is flown one step towards a stochastic combination of

[1] "Fitness" and "Cost" are used interchangeably in this text.
[2] Exact calculations may result in costly computations, which may increase the runtime of solution convergence.

the personal best and the neighborhood best position. Equation 1.1 shows a single particle's velocity and equation 1.2 shows the calculation for the new particle position.

$$v_{i+1} = v_i * w_i + C_1 * Rand_1() * (p_i - x_i) + C_2 * Rand_2() * (p_{nb} - x_i) \quad (1.1)$$

$$x_{i+1} = x_i + v_{i+1} \quad (1.2)$$

For the velocity calculation in Equation 1.1, $w_i$ provides an inertia to each particle and can be represented as a constant value, a linearly changing value, or a noisy value [7]. In the case of this work, the inertia value is noisy. The first component of the velocity calculation multiplies a constant $C_1$ to a random number between [0,1], and multiplies the random value by the distance from the personal best position to the current position. The second component of the velocity calculation multiplies the second constant $C_2$ by another random number between [0,1], and multiplies the new random value by the distance from the neighborhood best position and the current position. Equation 1.1 provides the behavior that large distances from personal best locations and neighborhood best locations have the greatest effect on velocity, but the effect is attributed with noise to reduce the likelihood of particles repeating search paths. The inertial part of the function allows particles to fly past *local optima* and *plateaus*.

PSO neighborhoods are defined by the number of neighbors each particle has. During each evaluation period, each particle's neighbors within the defined neighborhood are evaluated for the neighborhood best positions, and assigned to each particle. Naïve implementations of PSO provide a global best to the second component of Equation 1.1, which can also be considered similar to setting the number of neighbors to the population size minus one. The approach of searching for the global-best-only has been found to under-perform in comparison to neighborhood searches [18].

PSO repeats the fly and evaluate process until some satisfactory termination constraint is met. These constraints are defined as the maximum number of epochs to

run and a termination delta within a defined window. Each epoch is defined as a single iteration of a "fly" and an "evaluate" step. The maximum number of epochs serves a dual-purpose in preventing the program from running for infinity and for stopping the program at a sufficient approximation based on an empirical evaluation of the expected number of epochs to provide a sufficient result. Experiments may show that a sufficient result is expected after a certain number of epochs, allowing the user to decide on a trade-off between solution precision and runtime performance. The termination delta also serves a dual purpose by providing a minimum number of epochs to run defined by the window size and to terminate the process early once the solution precision becomes too fine. Similar to the experimentations with the maximum number of epochs, a minimum delta may be found that provides a trade-off between solution precision and runtime performance.

Figure 1.2 provides an example code snippet of the process function. The *while-loop* runs until some termination event and handles each particle. Each particle *flies*, *evaluates*, and *updates* its personal best and neighborhood best with each epoch. Upon termination, the global best is returned as the solution.

### 1.3.2 Human in the Swarm

The body of work shows many examples of human-swarm interaction: [19] provides interactive controls using selection and beacon to guide foraging swarms; [20] provides a combination of human interaction and swarm agents to solve various types of jigsaw puzzles; [21] uses humans as swarms and a GUI interface to answer poll questions; [22] manipulates the emotional state of humans through specific types of social media posts; [13] implements human controls for influencing fire-fighting robots; [15] provides human-interaction for selecting "utopia" points for trade-offs in fitness functions.

Much of the prior work in human-swarm interaction illustrates a desire to control the swarm through influence rather than direct and obvious control of individuals in the swarm. This work is inspired by this concept. The beacon aspect in [19] influences

```
Solution Pso::Process() {
    while(/*term event*/) {
        foreach(auto * p : particles) {
            p->Fly();                       // Fly Each particle
            p->Evaluate();                  // Evaluate "fitness"
            p->UpdatePersonalBest();        // Get best position
        }
        foreach(auto * p : particles) {
            p->UpdateNeighborhoodBest();  // Get neighbor best
        }
        /* Terminate ?? */
    } // while
    return GlobalBest(particles);           // Return global best
} //Pso::Run
```

Fig. 1.2. Generic example code for executing the PSO Process.

robots within a given range to either attract or repel robots to a new area. Despite the authors finding that selection results in better performance than beacon control, an approach is provided in [3] which implements a combination of beacons and zones for controlling the swarm. This work expands on [3] and provides updates to the prior work, resulting in better control of Zones and Beacons, user interface (UI) methods for manipulating pheromones, and better solution convergence.

### 1.3.3 Meta PSO

Meta PSO describes the use of PSO for optimizing the PSO problem itself. There are different ways to define Meta PSO. [23] discusses the use of PSO to optimize the parameters of the PSO itself, while using default parameters for the Meta PSO. [24] discusses implementing PSO with sub-swarms of PSO for training of a neural network to optimize PSO parameters and total number of neurons. Another consideration is to use PSO to optimize the fitness function. [25] discusses optimizing the weights of

the fitness function using Wmn-PSO. Methods other than PSO have been covered in finding the most appropriate fitness function for PSO. [26] selects several fitness functions and chooses the best performing one. [27] applies both fitness and reliability to a particle in order to estimate the fitness function. For the case of this work, an approach is made using PSO to optimize the fitness function weights to minimize variance and improve runtime costs. Due to the time-complexity cost of operating PSO on a PSO, Meta-PSO typically is not explored, resulting in fewer available works and a possible topic of further exploration.

## 1.4 Application of PSO

For the asset allocation problem, PSO is used to find the most optimal placement of assets among targets. Example representations of these assets can be considered as jammers covering EW targets or as radio towers receiving the best communication coverage. Placement of the assets are optimized in 3D space, frequency, and directional heading. Each asset is given a minimum sensitivity and maximum power level at the feedhorn and a fixed bandwidth. Targets are then assigned assets by considering the frequency bandwidth of the receiver, the power loss at the feedhorn, and whether or not line-of-sight (LOS) blockage is occurring. Power loss is modeled as free-space path loss [28, p.29], with future considerations for statistical multipath channel models and more complex propagation patterns. Equation 1.3 shows the power loss calculation $P_{Loss}$ based on straight-line distance in kilometers between the receiver and the transmitters, and frequency $f_{Mhz}$ of the transmission signal in megahertz (Mhz). Equation 1.4 shows the signal power at the receiver $P_{Rx}$ in dBm given the power transmitted $P_{Tx}$, the gain $G_l$ of the antenna, and the power loss equation.

$$P_{Loss}(dBi) = 20log_{10}(d_{km}) + 20log_{10}(f_{Mhz}) + 32.45 \tag{1.3}$$

$$P_{Rx}(dBm) = P_{Tx}(dBm) + 10log_{10}(G_l) - P_{Loss}(d_{km}, f_{Mhz}) \tag{1.4}$$

Signal assignments between transmitters and receivers are considered as follows. Each transmitting signal within the frequency range of the receiver are analyzed for assignment. The power at the receiver is calculated in dBm, converted to milliwatts, and added to the receiver's *total power received* for each non-blocked signal. Signals with the highest *power at the receiver* are considered the closest signals and are assigned to that asset. Once a signal is assigned, the power at the receiver and the particular transmitter assigned are recorded and remembered by the receiver. The signal assignments are then used for deriving the power and priority fitness components of the objective cost function, to be discussed in the near future.

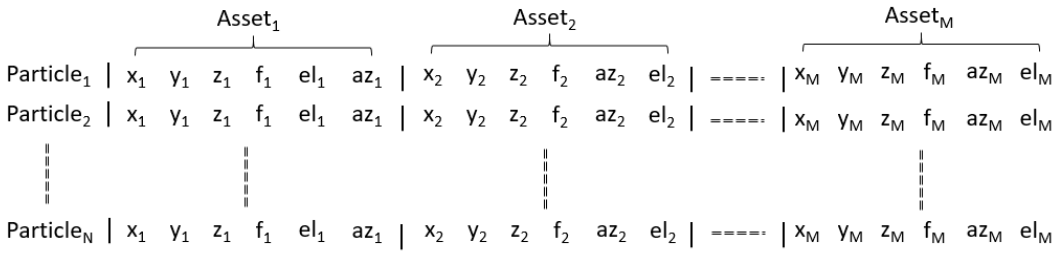



Fig. 1.3. Shown is the particle representation of the PSO. Each particle contains the spatial, frequency, azimuth, and elevation information for each asset.

Each particle in the PSO represents a particular solution. The solution consists of the set number of assets with an $x$, $y$, and $z$ location in $\mathbb{R}^3$, a center frequency within [4, 100] Mhz, and an azimuth and elevation angle in degrees. This representation is shown in Figure 1.3. As an example, if there are three assets to solve for, then the total dimension of the particle will be $3 \times (3+1+2) = 18$ dimensions. As each particle flies through the solution space, each element is individually flown with respect to the same element within other particles. In that sense, the solution space of particles is hyper-dimensional in $\mathbb{R}^{6N}$ where $N =$ assets.

[2]Future work will encompass observing more dynamic signal assignment algorithms.

$$Fitness_{overall} = \sum_{C \in Components} \alpha_C fitness_C \tag{1.5}$$

Each particular solution is judged by an objective fitness function, as shown in Equation 1.5, representing the weighted summation of five components: power, priority, spread, distance, and pheromone. Table 1.1 provides an outline of each component. The power is a summation or geometric mean of the received power of each asset. The priority is a weighted sum of all assigned signals, where the weight of each signal is defined by its priority. The spread is the minimum spatial spread among all the assets. The distance is the straight-line distance from the spatial centroid of the signals. The pheromone component implements human-in-the-swarm characteristics by providing a fitness based on the attraction or repelling of each asset from placed pheromones in spatiality.

Table 1.1.
The objective fitness function is calculated as a weighted sum of the following components:

| Components | Description |
|---|---|
| Power | Algebraic sum or geometric mean of power received across receivers. |
| Priority | Weighted sum of priorities of signal coverage. |
| Spread | Log of minimum spatial distance between receivers. |
| Distance | Inverse of distance of receivers from centroid |
| Pheromone | Attribution of pheromones to fitness cost |

## 1.5 Individual Contributions

The continuing chapters will outline the work embodied in this text. Each chapter gives a discussion about the concept, implementation, and results. Chapter 2 discusses the modifications made to the original code to implement stronger object-oriented programming practices, resulting in increased modularity, readability, and added functionality. Chapter 3 discusses human-swarm interaction and contributions made by adding pheromone-inspired beacons and zones. Chapter 4 discusses the

addition of directional antennas to the solution to increase the realism of solutions. Chapter 5 discusses how the fitness function weights are selected using a Meta-PSO method with a focus to reduce the variance, runtime, and violations across solutions. Chapter 6 provides analyzes the Meta-PSO results and proposes optimized weights. Chapter 7 provides a summary of research contributions, and Chapter 8 contains recommendations for future work

# 2. OBJECT ORIENTED PROGRAMMING MODIFICATIONS

## 2.1 Purpose

The initial structure of the PSO code was written using the object-oriented programming (OOP) paradigm, but much of the later additions were written using procedural concepts. To clarify, procedural programming relies on functions acting on data structures, whereas OOP is written with the primary intention of hiding the internal data via encapsulated data members while providing methods for accessing these members [29, 30]. Procedural programming relies on functions understanding the internal connections between data structures, whereas OOP hides internal data implementation and relies on message passing interfaces.

The concept of OOP is emphasized to reduce the complexity of implementation, reduce the possibility of developing stagnant and hard to develop code, and increase the productiveness of team-members and future developers. Much of the world-model is written in a procedural method, reducing the object-like nature of the receiver and transmitter simulations to data-structures. This introduces complexities and confusions in the project which lead to steep learning curves and long development times. With several simple modifications, the program is made more readable, stable, and modular.

## 2.2 Original Structure

Before discussing the modifications made, a quick recap of the existing structure of the program is necessary. Figure 2.1 shows the basic layout of the code struc-

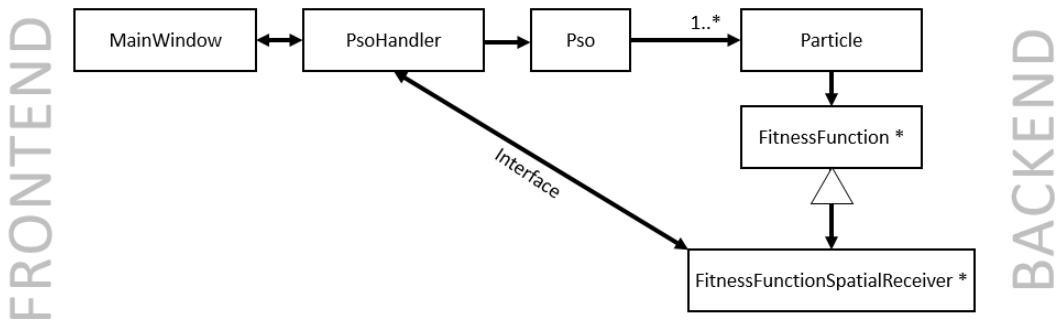



Fig. 2.1. The general flow of information goes from MainWindow (the frontend GUI of the program) to FitnessFunctionSpatialReceiver (the backend of the program).

ture in the program[1]. *MainWindow* provides the Graphical User Interface (GUI) functionality for interacting with the underlying program and implementing settings changes. The *PsoHandler* class serves as an interface between both the *Pso* and *FitnessFunctionSpatialReceiver (FFSR)* classes, and the *MainWindow* class. *Pso* is used to execute the PSO process by flying and evaluating each *Particle*. Each *Particle* contains a pointer to a class called *FitnessFunction*, which is an abstract class that must be inherited by a derived class that implements the *getCost()* method. *getCost()* is called by the *Evaluate()* function, shown in the PSO sample code from Figure 1.2. *FFSR* inherits *FitnessFunction* and provides the implementation for *getCost()*, as well as much of the programming logic required for representing the assets, signals, and solutions. Much of the prior modifications made to *FFSR* led to the class becoming a "superclass", resulting in reduced readability and becoming ultimately procedural. The rest of this chapter is devoted to describing the breaking down of *FFSR* to a more readable and modular class structure in order to implement more appropriate OOP practices.

First, take into consideration the structures used for our receivers, signals, assets, solutions, and signal assignments, shown in Figure 2.2. The receivers, signals, and

[1]Due to the size of these classes, it would be unfeasible to display the entire UML of each here.

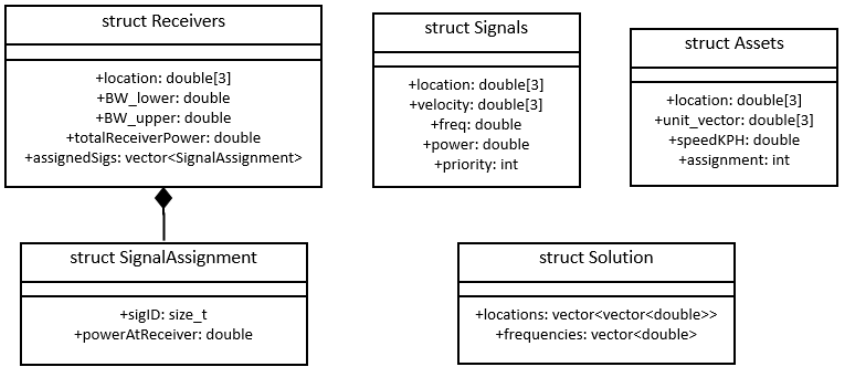



Fig. 2.2. Original struct implementaions for receivers, signals, signal assignments, and solutions.

solutions are given structs that share concepts of frequency and location. Furthermore, solution stores its concept of location differently than signal, resulting in the possibility of confusion and bug introduction. The locations within the array signify $x$, $y$, and $z$, but these array locations must be understood in advance by the programmer. This can become a problem in other cases, where the y and z axis are flipped for 3D display. Another problem to be noted is that both signals and assets have the concept of velocity, but both use different implementations for representing velocity. One more problem is that receivers, signals, and signal assignments have a concept of power, but what's not known is the unit that these powers are stored in, resulting in the potential for mismatching decibels, watts, and milliwatts.

## 2.3 PhysicalObject Class

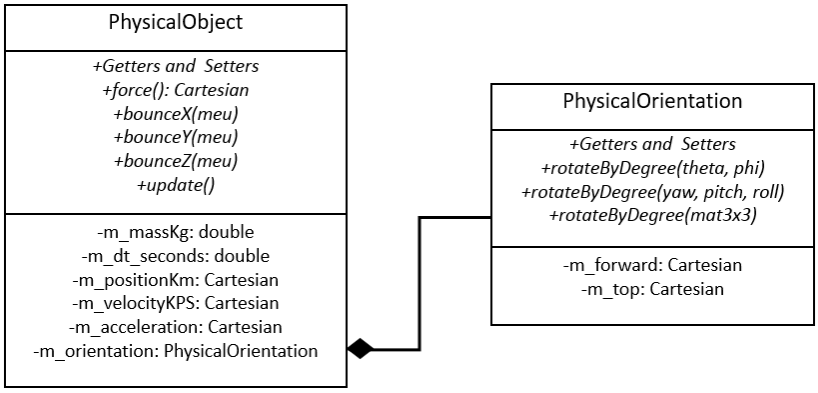



Fig. 2.3. PhysicalObject and PhysicalOrientation UML. All vectors for position and velocity are replaced by PhysicalObject objects.

From these observations, work is implemented to develop several classes capable of unifying spatial, frequency, and power concepts. Figure 2.3 shows the Unified Modeling Language (UML) for two classes: *PhysicalObject* and *PhysicalOrientation*. *PhysicalObject* stores physics information for an object, with respect for mass, position, velocity, acceleration, and orientation. The position, velocity, and acceleration are stored as *Cartesian* objects, shown in Figure 2.4, which provide unifying methods for storing spatial data as vectors, while providing access as *x()*, *y()*, and *z()*. This maintains the original intention of vector optimization for positional data manipulation, while maintaining a uniform representation of spatiality for all objects in the program. Furthermore, *Spherical* and *Cylindrical* classes are created to handle conversions between the three coordinate systems.

## 2.4 XcvrObject Class

*PhysicalObject* is the class that handles the spatial location and movement of an object, but the objects for receivers, assets, and solutions also requires frequency,

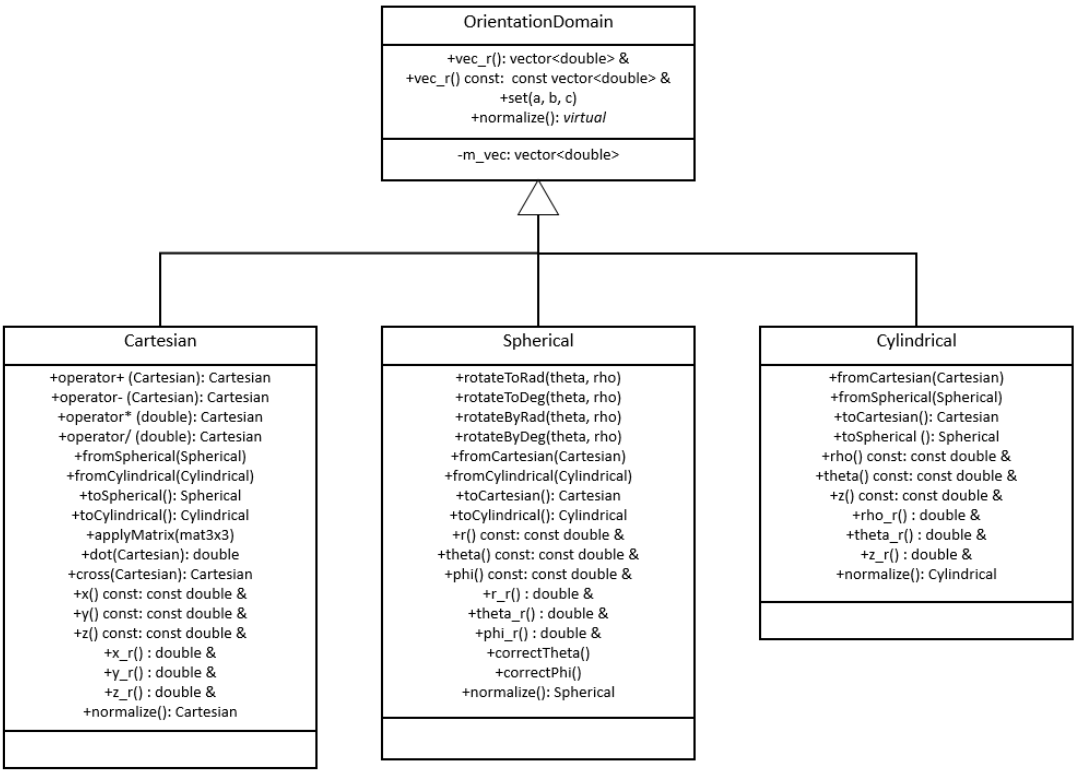



Fig. 2.4. UML for Geometric objects. *OrientationDomain* is an abstract class containing a private vector member, which stores a three-element vector. *Cartesian*, *Spherical*, and *Cylindrical* inherits from *OrientationDomain* to implement the different geometric domains.

power, and assignment capabilities. Much of this is handled in *FFSR* functionally, but can also be implemented more appropriately using OOP techniques. The *XcvrObject*, named to symbolize both the transmitter and receiver aspect of a transceiver and shown in Figure 2.5, is a class implemented to encapsulate the transmitter and receiver concepts defined in *FFSR*, while maintaining the original functionality of the program. Much of the functionality within *FFSR* is moved from *FFSR* into *XcvrObject* as static functions or object members. *XcvrObject* contains both transmitter and receiver center frequencies, powers, and bandwidths, as well as signal assignments for receivers. Calculations for free-space-loss, distance-to-horizon, to and from dBm and mW are statically implemented via *XcvrObject*, removing the responsibility from

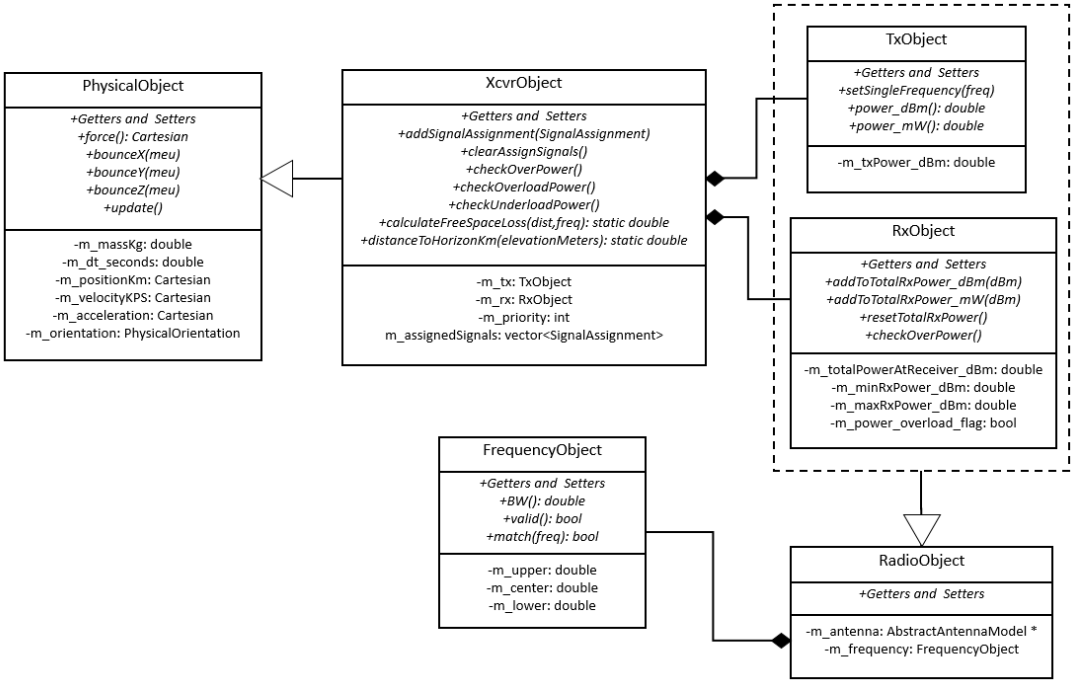



Fig. 2.5. UML for *XcvrObject*, *RadioObject*, *RxObject*, *TxObject*, and *FrequencyObject*. Each class shares responsibility for implementing RF propagation and attaching it to a physical object. Receivers and signals are defined as instances of *XcvrObject*.

*FFSR* and associating these functions with a RF-related class. Calculations for checking over-power and under-power of signal assignments is also encapsulated within the *XcvrObject* as an object method, resulting in a more compact and organized method for polling receivers for power constraints. *XcvrObject* inherits *PhysicalObject*, resulting in the ability to define the position and RF properties within a single object in a uniform and consistent manner, resulting in repeatable and stable code while maintaining fast convergence times.

## 2.5 TerrainData Class

Much like the signal and receiver objects, terrain handling is promoted to an object called *TerrainData* shown in Figure 2.6. In prior work, terrain elevation was

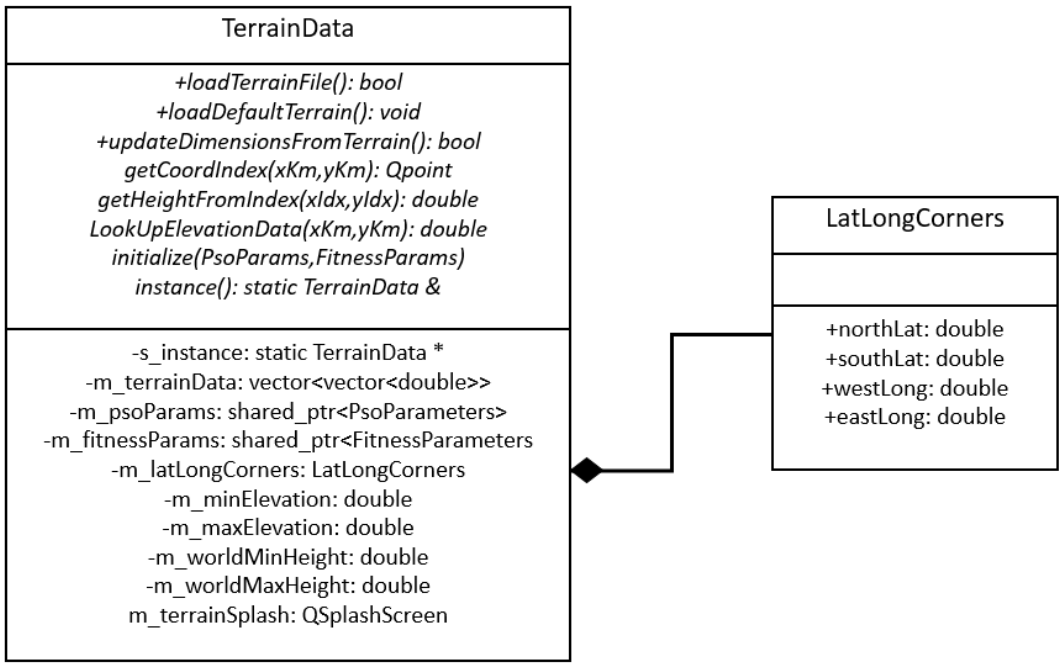



Fig. 2.6. UML for *TerrainData* and *LatLongCorners*. *TerrainData* stores the 2D height map for terrain elevation data, and encapsulates all terrain-related aspects.

stored in a 2D vector matrix of type double, which allowed for quick access to terrain data via a simple indexed lookup. The problem with this kind of setup is that the programmer must understand which index contains what information, what the resolution of that data is, what units it is in, and provide many different functions for accessing and modifying this data. Many of these functions were implemented as members of *FFSR*, resulting in clutter and shadowing the intentions of *FFSR*. Much of this is easily encapsulated within *TerrainData* by transferring ownership of that 2D vector matrix to *TerrainData* as a private member, and providing methods for loading, accessing, and translating elevation points when necessary. *TerrainData* is implemented as a Singleton, allowing static access to the terrain object through a call to *instance()*. By moving all of the terrain-related concepts from *FFSR* to *TerrainData*, access to terrain information is more easily understood. This resulted

in a simple check for line-of-sight (LOS) blockage by scanning along the terrain from point to point and verifying elevation points between two objects.

The advantages of using OOP practices to implement the receivers, transmitters, solutions, signal-assignments, and terrain as objects, rather than data structures, results in making the code much simpler to not only read and understand, but to also modify. After these modifications were made, the task for implementing antenna models was made quick and simple, resulting in the initial implementation for antenna models to be written and tested in under a week. The implementation of antenna models is discussed further in Chapter 4.

## 2.6 Statistics Collection

Statistics are collected after the calculation of each solution. The collected statistics provide a method to measure the mean, variance, standard deviation, and confidence interval of each asset's position, frequency, and heading for each run. Furthermore, statistics are collected on the output of each fitness calculation for each fitness component for each epoch. Figure 2.7 shows the UML for the statistics container.

### 2.6.1 StatObject

An object is created for calculating all statistics, called *StatObject*. This object holds a circular queue of data points with a set limit. In this project, the limit is set to 2000 data points by default. Data is entered into the *StatObject* by calling *add_val()*, which takes a *double* value $x$. This value is then added to two sums, shown in Equations 2.1 and 2.2,

$$Sum = Sum + x \tag{2.1}$$

$$Sum_{sqr} = Sum_{sqr} + x^2 \tag{2.2}$$

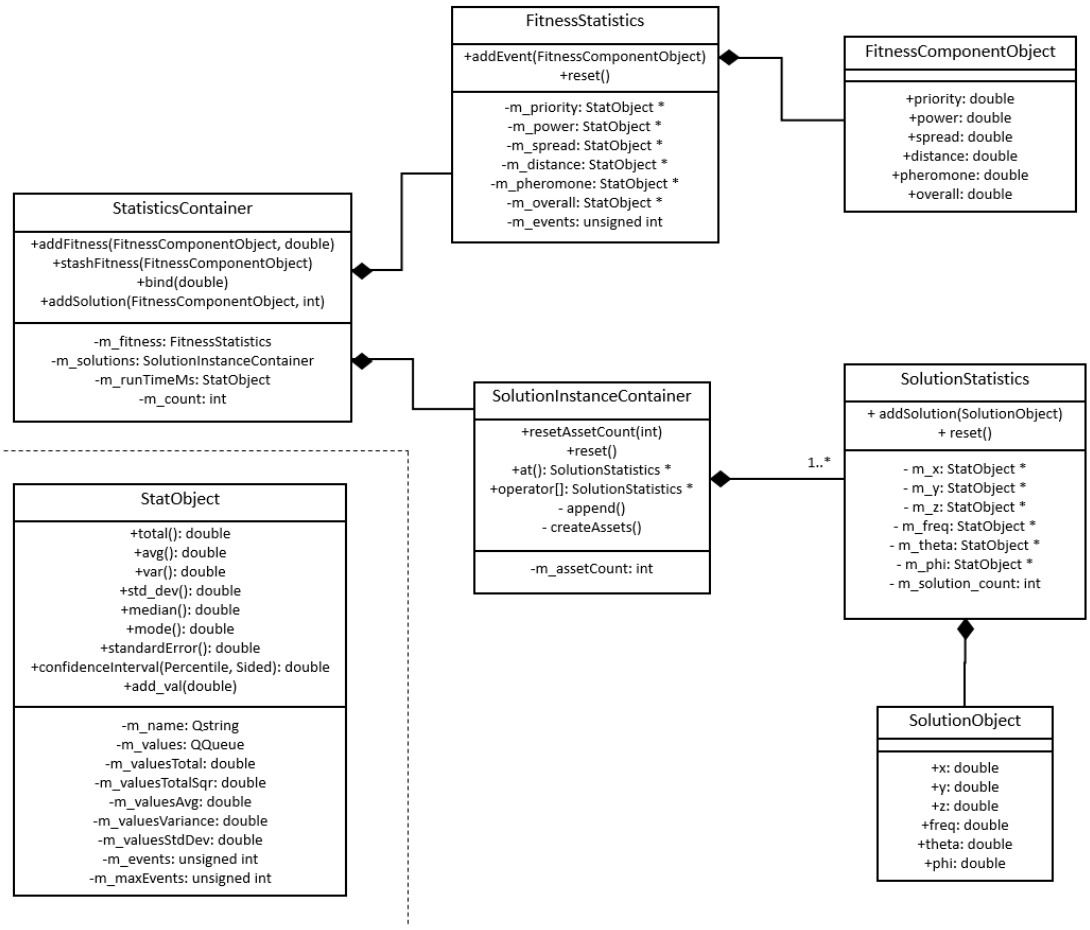



Fig. 2.7. UML for StatisticsContainer

These two values are used for calculating the sample mean, variance, and standard deviation. Equations 2.3, 2.4, and 2.5 show how mean, variance, and standard deviation are calculated, respectively.

$$\bar{x} = \frac{Sum}{N} \tag{2.3}$$

$$s_x^2 = \frac{Sum_{sqr} - \frac{Sum^2}{N}}{N-1} \tag{2.4}$$

$$s_x = \sqrt{s_x^2} \tag{2.5}$$

Equation 2.4 is a fast way to calculate the sample variance, which is shown in Equation 2.6.

$$s_x^2 = (\bar{x^2}) - (\bar{x})^2 = \frac{\sum_{i=1}^{N} x_i^2 - (\frac{\sum_{i=1}^{N} x_i}{N})^2}{N-1} \tag{2.6}$$

The confidence interval is calculated using the Student's t-distribution following Equation 2.7. Student's t-distribution is calculated by a lookup table, which is accessed by percentile, degrees of freedom, and sidedness. By default, the function for calculating the confidence interval is set to find the double-sided 95% percentile, but these can be changed by the enumerations, shown in Figure 2.8.

$$CI_{95\%} = \bar{x} + t_{95\%,N-1}\frac{s_x}{\sqrt{N}} \tag{2.7}$$

```cpp
class StudentTDistribution
{
public:

    enum Percentile {
        PERCENTILE_50,
        PERCENTILE_60,
        PERCENTILE_70,
        PERCENTILE_75,
        PERCENTILE_80,
        PERCENTILE_85,
        PERCENTILE_90,
        PERCENTILE_95,
        PERCENTILE_97_5,
        PERCENTILE_98,
        PERCENTILE_99,
        PERCENTILE_99_5,
        PERCENTILE_99_75,
        PERCENTILE_99_80,
        PERCENTILE_99_90,
        PERCENTILE_99_95
    };

    enum Sided {
        ONE_SIDED,
        TWO_SIDED
    };

    static double confidenceInterval(
            const double & sample_variance, const int & samples,
            const Percentile & p = PERCENTILE_95, const Sided & s = TWO_SIDED);

    static const double & coeff(const int &degreesOfFreedom, const Percentile & p, const Sided & s);
    static const double & coeffOneSided(const int & degreesOfFreedom, const Percentile & p);
    static const double & coeffTwoSided(const int & degreesOfFreedom, const Percentile & p);

private:
    StudentTDistribution() = delete;

    static std::vector<std::vector<double>> s_coeff;
    static size_t getIdxFromDegreesOfFreedom(const int & degreesOfFreedom);

};

```

Fig. 2.8. Shown is the class for Student's t-distribution called *StudentTDistribution*, which is used for handling the lookup of the distribution coefficients.

### 2.6.2 Statistics Containers

There is a temporal separation between the statistics for collecting the fitness component values and the statistics for collecting the solution values. The fitness components are to be collected for each particle and each generation, but the solution values are collected after the completion of a full solution. This temporal separation is necessary because both types of statistics collection are telling different stories. The fitness values are giving information for which fitness components are the most

influential throughout hunting for a solution, whereas the solution values are giving information for how similar each solution is from each other.

The separation of these two concepts within the same class are handled through two different function calls. For each call to *getCost()*, the fitness component values are stored to the FitnessStatistics object. As a reminder, *getCost()* is called for each particle and each epoch, so the fitness statistics are aggregates of all the particles for the entire epoch. Collecting this data is valuable for determining which components are dominating the fitness function.

Once a final solution is obtained, the runtime statistics, in milliseconds, is updated in the *StatisticsContainer*, and the solution statistics are passed to the *SolutionInstanceContainer*. *SolutionInstanceContainer* maintains the list of *SolutionStatistics*, where each element in the list is an asset. *SolutionStatistics* tracks the spatial, frequency, and heading statistics for each asset. Two objects shown in Figure 2.7, *FitnessComponentObject* and *SolutionObject*, are basic structs used for passing aggregated instance information.

### 2.6.3 K-Means Clustering of Solution Statistics

As discussed in [4], each PSO solution is not a one-for-one representation of assets. Across different solutions, different assets may associate well with each other. For example, asset 0 of solution 0 may better represent asset 3 of solution 1. In this sense, we can consider each asset across all solutions as belonging to a member of a cluster. K-means clustering, with a known $k$ equal to the number of assets, is used to reassign the asset statistics in *SolutionInstanceContainer*. K-means clustering is a method which finds bins for the classification of points that minimizes the distance between each point in the cluster and the cluster's centroid. The result of k-means clustering is to gain a more accurate statistic representation for each asset.

K-means is executed on the *SolutionInstanceContainer* by implementing a temporary copy of the container. This copy is assigned the median of each asset's collected

$x$, $y$, and $z$ statistics as a starting point. The Euclidean squared distance from this point is calculated for all assets within the container, and each asset is given an assignment to the closest cluster. Once all assets are assigned a new cluster, the cluster's centroid is updated by calculating the mean value for $x$, $y$, and $z$ in the copy list. This event continues until a minimum count of three iterations is reached and the distance is no longer getting better.

K-Means is called via a function named *StatisticsContainer::refactorSolution()*. This function is connected to a menu option in MainWindow, and is used for repeated runs of the PSO, Meta-PSO, and analysis of fitness weights.

# 3. HUMAN IN THE SWARM

## 3.1 Purpose

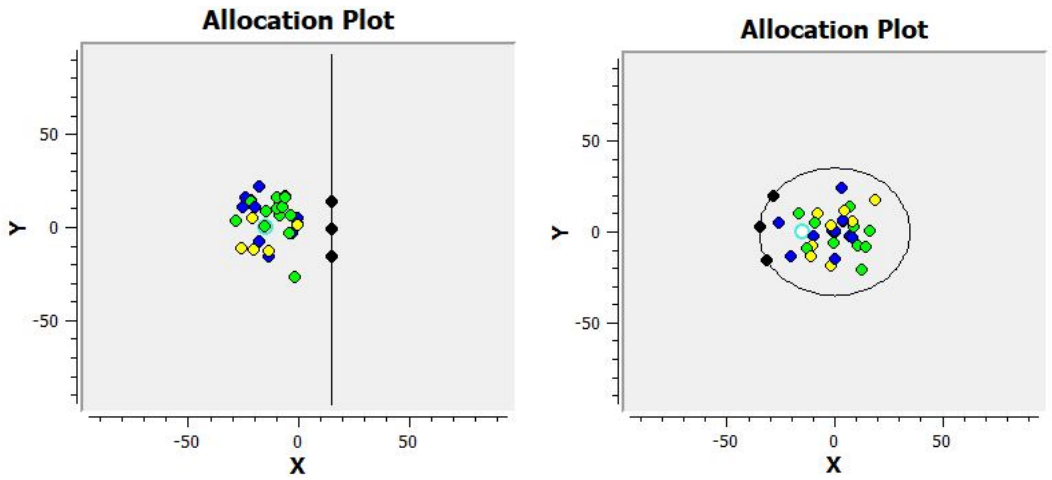
So far, PSO has been used to provide solutions for dynamic problems without much consideration for the concept of "good" or "bad" areas. It may be possible that the situation changes and that the solution must take into account areas of the field that are preferred or off-limits. A human may be able to quickly inform the PSO of these areas, but a method is required for establishing these boundaries that is both intuitive and quick to process in real-time. Furthermore, the DOD has a strong command-and-control philosophy, requiring that automation must have human oversight.

Before discussing human-swarm interaction, it is important to make a distinction about swarms and PSO in this work. It should be considered that this project is optimizing a *swarm of assets* using a *swarm optimization algorithm*. In a sense, there are two separate swarms with completely different characteristics and qualities. The swarm of assets are the receivers being placed to provide coverage of the EW field, whereas the PSO is a separate, centralized swarm that is solving this problem. Each particle in the PSO has an opinion of what the final asset swarm should look like. In the context of this text, human-swarm interaction is defined as the interaction of humans with the asset swarm, and not the PSO.

## 3.2 Prior Work

Prior work in [2] provides an initial implementation to a keep-away line, shown in figure 3.1(a). This line works by providing a penalty for any solution that results in assets which cross to the left of the line. The penalty is a constant penalty multiplied

to the overall fitness. Another implementation provided by [2] is a keep-away circle that ensures assets are generated outside of the circle, shown in Figure 3.1(b). The keep-away circle behaves similarly to the keep-away line by providing a constant penalty for all particles in violation.



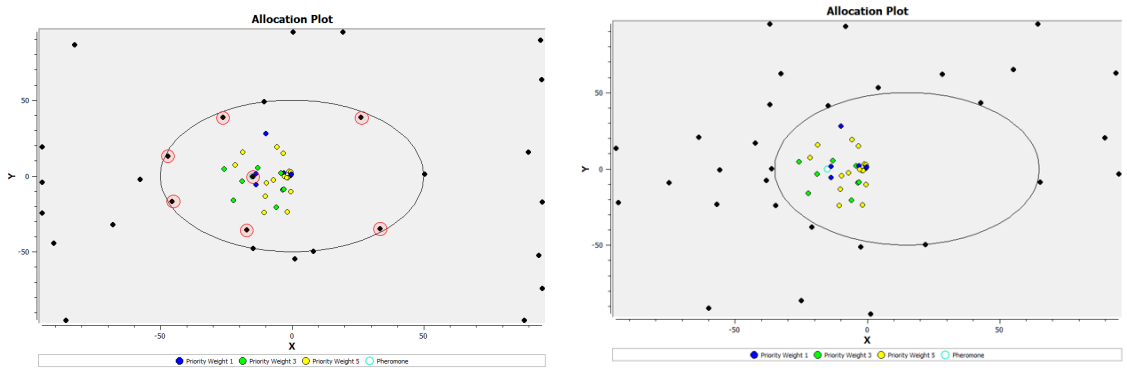

(a) Keep-Away Line: x=15 km

(b) Keep-Away Circle: radius=35 km

Fig. 3.1. (a) shows the keep-away line, illustrated with a black verticle line; all assets must stay on the right of the line. (b) the keep-away circle, illustrated with the black circle; all assets must stay outside of the circle.

The trade-off for such implementations is that [2] provides a quick and efficient way for determining a "keep-away" zone, but does so at the expense of a fitness gradient. By forgoing a gradient, the algorithm relies on the PSO being capable of providing at least one potential solution within the short steps of epochs that can observe the safe zone. Without a gradient guiding the solution to the zone, and if such a solution is rare, the PSO may converge on a solution agnostic to the zone and never be guided to it. It has been observed that too strict of constraints on a PSO problem results in a highly sparse solution space, resulting in a higher probability of missing the global best result. In other words, too strict of constraints on a PSO problem can result in certain constraints being ignored while searching for better solutions, simply because

it is too low of a probability for the PSO to ever witness better states that meet the constraints. This is shown in Figure 3.2(a) where too many assets results in violations of the keep-away circle. Of the 200 particles used for this example, neither of the particles were able to converge on a solution that guided those 7 assets outside of the zone. It is necessary to devise a method that is able to handle this strict-constraint problem, while maintaining the same or acceptable computational complexity. Figure 3.2(b) shows the usage of a pheromone-based concept under the same conditions to guide the solution using a soft-step function, shortly to be explained.




(a) Legacy Keep-Away Circle

(b) Pheromone Keep-Away Circle

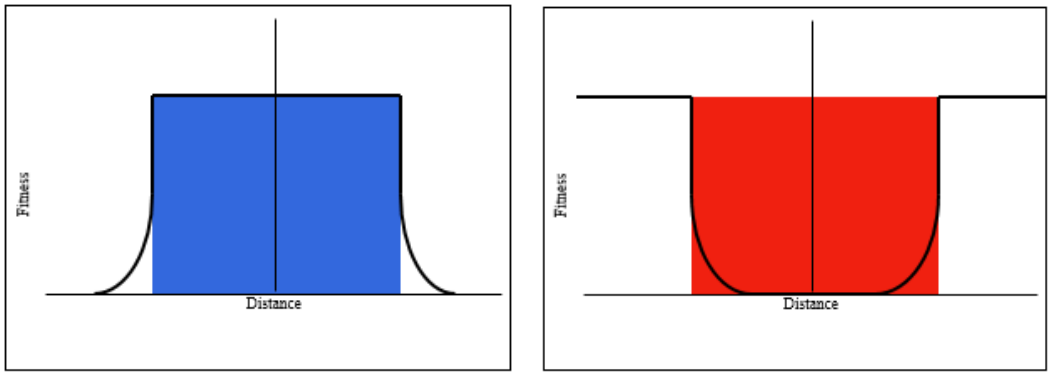
Fig. 3.2. Example of keep-away circles of 50 km radius in both legacy (a) and pheromones (b). 30 assets are optimized for placement: 7 violate the keep-away circle penalty in (a); 0 violate the keep-away circle penalty in (b).

## 3.3 Concept

A concept for human interaction with PSO convergence [3] is introduced, which provides a pheromone-based, mixed-discretized method that handles the strict-constraint problem while maintaining similar run-time results as [2]. Furthermore, procedures are implemented for allowing the user to interact with the 3D visualization for placement and movement of boundaries. Pheromones are natural chemical reactions in

animals, which provoke a response for sexual behavior, food acquisition, and enemy avoidance. Animals may emit pheromones when sexually aroused to attract partners and ants may lay pheromones to guide other ants towards locations with food. In this sense, pheromones disperse and travel, guiding the organism much like magnetism or gravity towards the source or the sink.

In the "pheromones" developed for this project, a soft-step function is implemented to guide the convergence of the PSO to a boundary or point. A weighted fitness component is added to the original project's overall fitness cost, and fitness is attributed based on distance away from the placed pheromone. The fitness cost follows an exponential equation that either increases or reduces when an object becomes closer or further from the pheromone, respectively. The pheromones are modeled in two ways: either with a radius as a "Zone," or without a radius as a "Beacon."




(a) Attract Zone (b) Repel Zone

Fig. 3.3. Shown are the concepts representing pheromone zones. (a) shows the concept for fitness representation of the attracting pheromone zone. (b) shows the repelling pheromone zone.

Zone pheromones provide a keep-in or keep-out zone for the assets with an associated strength level, as shown in Figure 3.3. For example, the keep-in zone provides a very low fitness cost for assets that are far away. This component cost increases exponentially within the range of [0,0.25] until reaching the radius. Once the asset

is inside of the pheromone's zone, the fitness is set to a constant 1.0. This allows for a mixed-discrete fitness function that not only attracts assets to keep-in zones, but provides enough incentive for assets to stay inside of the keep-in zones. Similar to keep-in zones, the keep-out zones behave in reverse by pushing assets away from the centroid of the zone, and applying constant fitness once outside. The conceptual equations for the zones are shown in Equations 3.1 and 3.2. The significance of 283 in the denominator of the exponent is for the diagonal distance between the corners of a $200 \times 200$ $km^2$ map. The 5 multiplier signifies a near-zero value for distances near the border of the terrain map. 200 $km$ is the expected maximum distance imposed on the simulation and provides a good slope for the fitness function component.

$$Zone_{KeepIn}(d_{km}) = \begin{cases} 0.25(1 - e^{-\frac{5d_{km}}{283.0}}), & \text{if } d_{km} > radius \\ 1.0, & \text{if } d_{km} \leq radius \end{cases} \tag{3.1}$$

$$Zone_{KeepOut}(d_{km}) = \begin{cases} 0.25(e^{-\frac{5d_{km}}{283.0}}), & \text{if } d_{km} \leq radius \\ 1.0, & \text{if } d_{km} > radius \end{cases} \tag{3.2}$$

Beacon pheromones, on the other hand, do not provide a zone for keeping in or keeping out. The fitness for the Beacon pheromone is a mostly continuous spike in the field for objects near or far from the Beacon. Unlike [19], beacons are global and are attributed to all assets and not just those within a radius of the beacon. Equations 3.3 and 3.4 show the equations for beacons.

$$Beacon_{Attract} = 1 - e^{-\frac{5d_{km}}{283.0}} \tag{3.3}$$

$$Beacon_{Repel} = e^{-\frac{5d_{km}}{283.0}} \tag{3.4}$$

Multiple pheromones are added together, where Zone pheromones are weighted 1,000 more than Beacon pheromones. The addition of the zones introduces a dynamic

field of pheromones, which can be manipulated by the human in various configurations for different responses. One ideal implementation is to mix the Zone keep-away pheromone with an attracting beacon, which allows for more dynamic control of attracting assets to coverage of possible dangerous locations, while maintaining appropriate distances defined by the human.

## 3.4 Implementation

Implementation has been split into two subsections: calculation and placement. Calculation discusses the implementation for the calculation of pheromone effects on assets. Placement discusses the implementation for the user to place and manipulate pheromones.

### 3.4.1 Calculation

Pheromones are implemented as a list of objects called *PheromoneElements*, which the user can add, remove, and move dynamically. The UML for *PheromoneElements* and the *PheromoneList* are shown in Figure 3.4. *PheromoneList* is statically defined as a singleton design pattern for simple access with a vector of *PheromoneElements*, which can be accessed publicly. The primary purpose of *PheromoneList* is to provide simple methods to calculate the pheromone cost and validate potential solutions. *PheromoneElement* serves the purpose of encapsulating the calculation of the cost of an affected receiver by defining the particular pheromone attribute and shape and selecting the correct cost function. The function *getCost()* takes the *PhysicalObject* as a parameter, which is the base of *XcvrObject*, and calculates the cost based on the distance between the *PheromoneElement* and the receiver, and defined settings.

The calculation of the pheromone cost for each *PheromoneElement* is determined by the *PheromoneAttribute* and *PheromoneShape* enumerations, defined and assigned for each *PheromoneElement*. Figure 3.4 shows the potential values for both enumerations. *PheromoneAttribute* defines whether the *PheromoneElement* is a Beacon or a

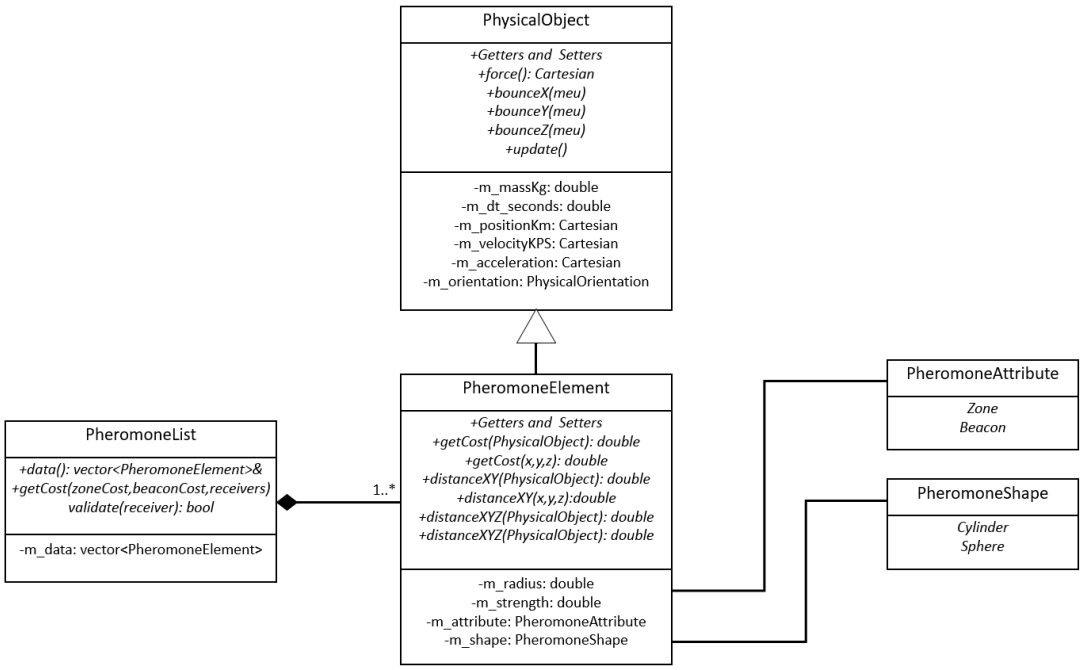



Fig. 3.4. The UML for PheromoneElement and PheromoneList. PheromoneList contains a list of PheromoneElements and provides methods for validating and getting the cost of solutions.

Zone pheromone. *PheromoneShape* is a placeholder for different shapes, where only *Cylinder* has been implemented.

Table 3.1.
The 4th-order polynomial approximations used for pheromone fitness.

| Action | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
|---|---|---|---|---|---|
| Attract | $4.423 \times 10^{-10}$ | $-3.564 \times 10^{-7}$ | $1.102 \times 10^{-4}$ | $-1.610 \times 10^{-2}$ | $9.871 \times 10^{-1}$ |
| Repel | $-4.423 \times 10^{-10}$ | $3.564 \times 10^{-7}$ | $-1.102 \times 10^{-4}$ | $1.610 \times 10^{-2}$ | $1.291 \times 10^{-1}$ |

Zones and Beacons are calculated using a polynomial approximation to the fitness function in Equations 3.3 and 3.4. A polynomial equation is used due to the computational complexity cost of calculating the *std::exp()* provided by the standard template library (STL) math library. The equation used for the polynomial approximation is a

$4^{th}$-order polynomial equation; the coefficients for the polynomial equation is shown in Table 3.1. The purpose for choosing the $4^{th}$-order approximation is shown in Figure 3.5, where the $4^{th}$-order approximation is shown to be faster, and Figure 3.6 shows the approximation to be a close representation of the exponential function within the range of [0,283] with a correlation coefficient of R=0.999794.

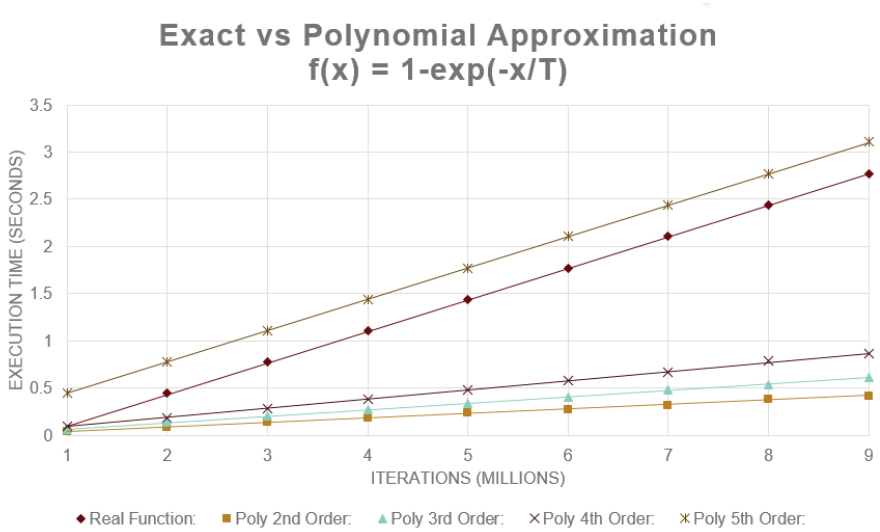



Fig. 3.5. Shown is the time-cost for execution of millions of iterations of the std::exp() function, in comparison to its polynomial approximations. The 4th-order approximation offers the best trade-off.

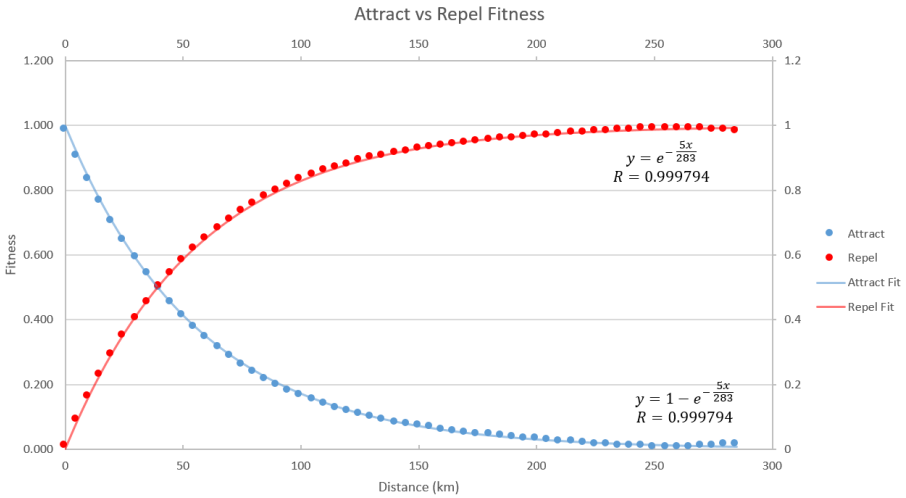



Fig. 3.6. The graph shows the polynomial approximation of the exponential and logistics functions for attracting and repelling pheromones.

### 3.4.2 Manipulation

Manipulation of pheromones is implemented in two ways: table and GUI mouse clicks. The table offers an easy way to define each pheromone's placement and settings, while showing the existing pheromones. Figure 3.7 shows the implementation for the pheromone table. Pheromones can be added by pressing the "Add" button, deleted by checking the "Delete [ ]" checkbox and pressing delete, and modified. Modifications are made for X and Y placement, as well as radius, strength, and attribute. Attribute controls Beacon or Zone type, and strength determines whether attracting or repelling. Attracting pheromones are negative strength, and repelling pheromones are positive. Both "time" and "shape" are not implemented.

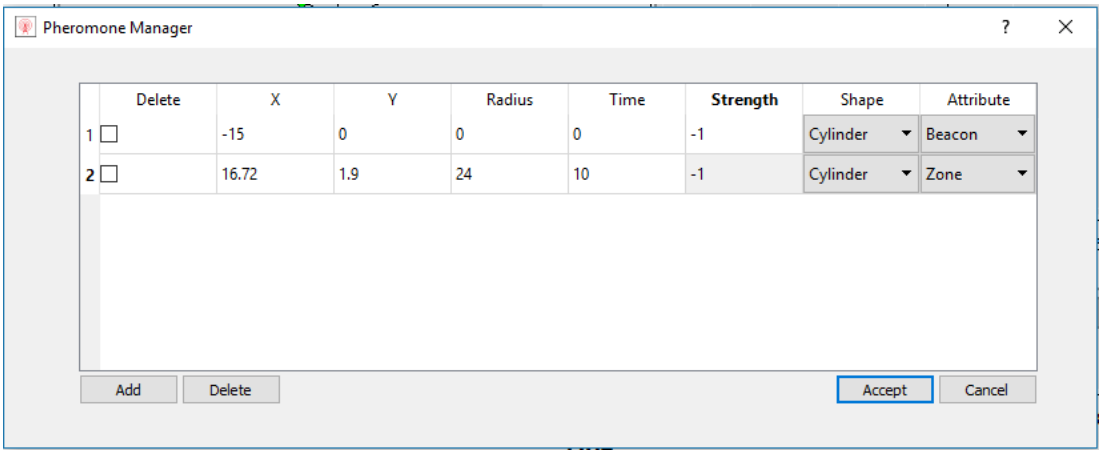
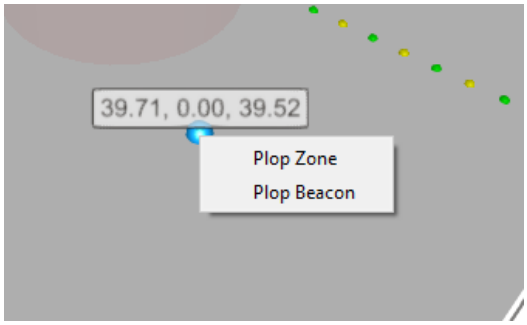


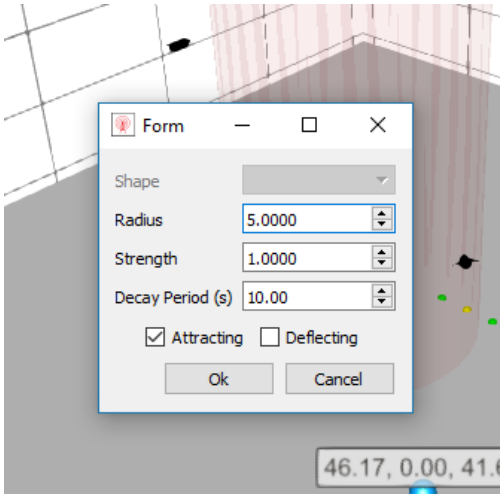

Fig. 3.7. GUI for Pheromone Management Table

Pheromones can be manipulated from the 3D window through mouse clicks, as shown in Figure 3.8. Right-clicking on the surface triggers a response from Qt Data Visualization that gets the surface's location in $\mathbb{R}^3$ in kilometers. Using this information, the context-menu asks the user whether to "plop" a Zone or a Beacon. Figure 3.8(b) shows the controls for selecting the Zone, which provide implementations for changing the shape, radius, strength, decay period, and whether or not the pheromone is attracting or repelling (deflecting). The decay period originally was meant to pro-

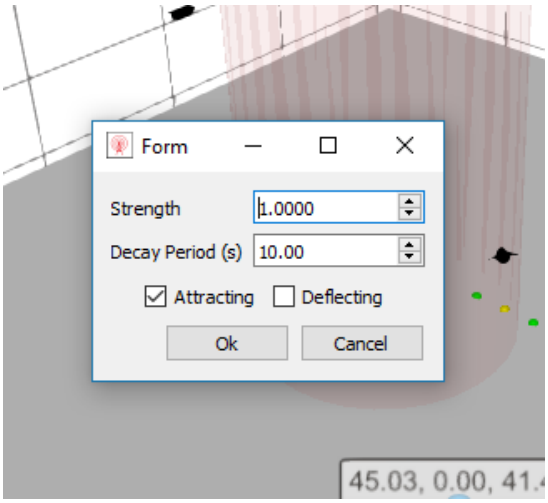
vide temporary time frames for the pheromones, but was removed and has no affect. Figure 3.8(c) shows the controls for the Beacon pheromone, where radius and shape selection has been removed.




(a) Right-click Context Menu



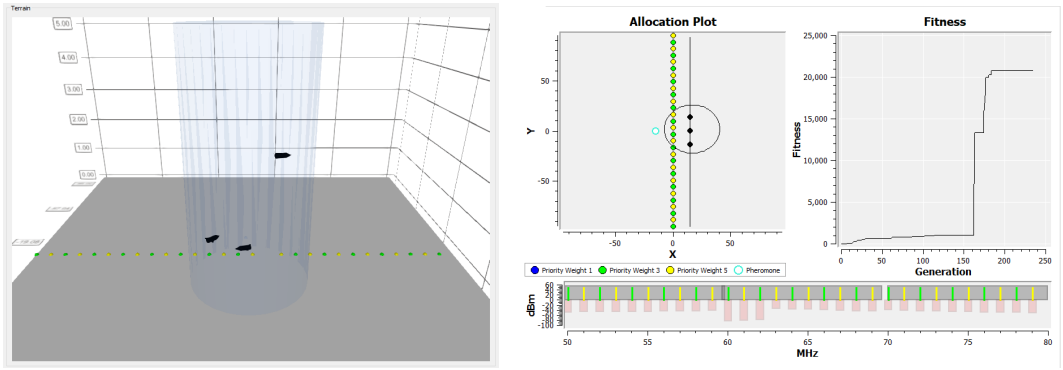

(b) Zone Selected




(c) Beacon Selected

Fig. 3.8. Shown are the mouse controls for placing pheromones. (a) shows the context-menu for right-clicking on the 3D surface plot. (b) shows the selection of "Plop Zone". (c) shows the selection of "Plop Beacon".

Furthermore, pheromones can be moved through a similar process as placement by left-clicking on the pheromone through the 3D window and moving the mouse. Clicking on the window provokes a response, which requests which *QCustom3DObject* is selected. A controller was implemented which controls which and how pheromones are drawn to the 3D window. This controller handles the mouse-clicks and manages the movement of pheromones.




(a) Keep-In Zone 3D

(b) Keep-In Zone 2D

Fig. 3.9. An example of the keep-in zone being used for a straight line. The assets are forced inside the keep-in zone, where they optimize to cover all transmitters. (a) shows the 3D representation as a blue transparent cylinder. (b) shows the 2D representation as a circle around the 3 black assets in the "Allocation Plot".

Figure 3.9 shows an example of the attracting keep-in zone for a straight-line transmitter placement problem. The assets are able to converge on a solution that covers all transmitters, restricting all assets inside of the keep-in zone. Figure 3.10 shows the same problem using a keep-away zone as a red cylinder. The assets are now shown to converge to a solution outside of the zone.

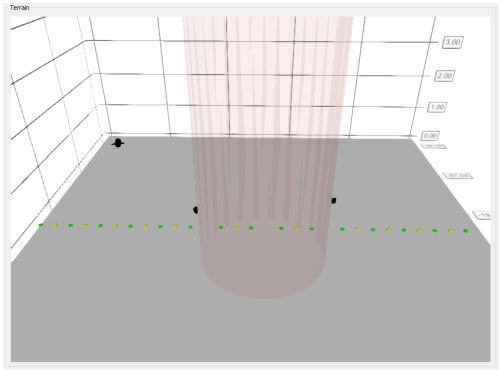
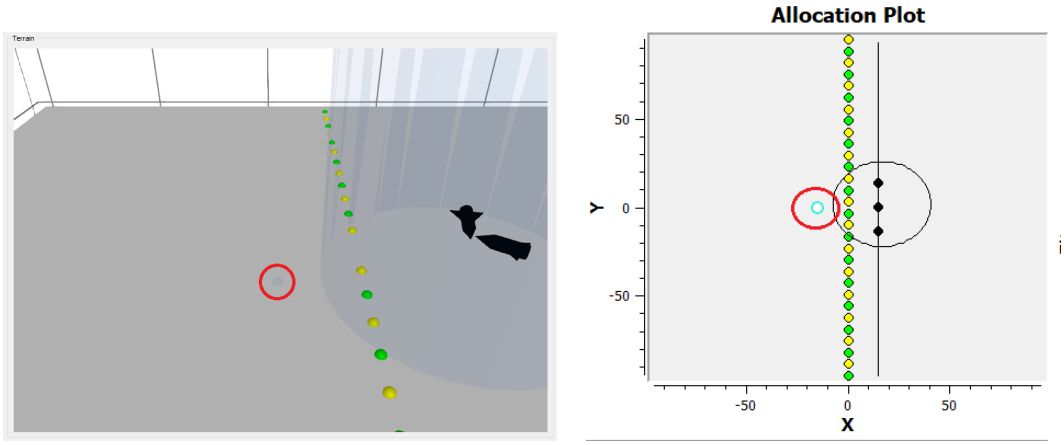



Fig. 3.10. Shown is the keep-out zone in red.

In both solutions, an attracting Beacon is used to attract the assets to a centralized point, shown in Figure 3.11. Attracting Beacons are shown as blue transparent balls in the 3D plot and cyan circles in the 2D plot. Repelling Beacons are shown as red transparent balls in the 3D plot, and the same cyan circle in the 2D plot. It is shown that beacons do not overpower the keep-away line, nor the keep-away pheromone zones.




(a) Attracting Beacon 3D

(b) Attracting Beacon 2D

Fig. 3.11. The Beacon is being emphasized by red circles. (a) is showing the small transparent blue dot circled in red. (b) is showing the cyan circle circled in red. The red circles are not part of the actual program.

# 4. DIRECTIONAL ANTENNA

## 4.1 Purpose

Up until this point, this project has assumed the usage of an isotropic antenna with unity gain in all directions. Due to the complex nature of RF propagation, the realization of isotropic antennas is considered impossible and only used as a reference [31]. Advanced methods of antenna pattern modeling and RF propagation have allowed for a more realistic model of the RF battlefield environment. The usage of directional antennas is just as relevant as isotropic antennas. Therefore, it is necessary to consider the implementation of solutions with directional antenna models. There has been peaked interest in determining solutions to the asset allocation problem with consideration for the effects of directional antenna gains. In this chapter, an implementation of such antenna types is accomplished and tested, proving the effectiveness of using PSO for use with a basic model of directional antenna, and sparking interest in continued research with more advanced considerations.

## 4.2 Concept

Directional antenna are characterized by their radiation patterns with respect to the main-lobe, side-lobes, and back-lobe. The main-lobe is the intended transmission space of the signal, which is characterized by the beam-width. Beam-width is determined by calculating the angle between the two points marking the -3 dB half-power drop-off [32]. The side-lobes and back-lobe are the lobes which fall outside of the beam-width, and directly in the opposite direction of the main-lobe, respectively. These lobes are considered the unintentional byproduct of the antenna design and are not typically used for communication purposes, with the exception of the maximum

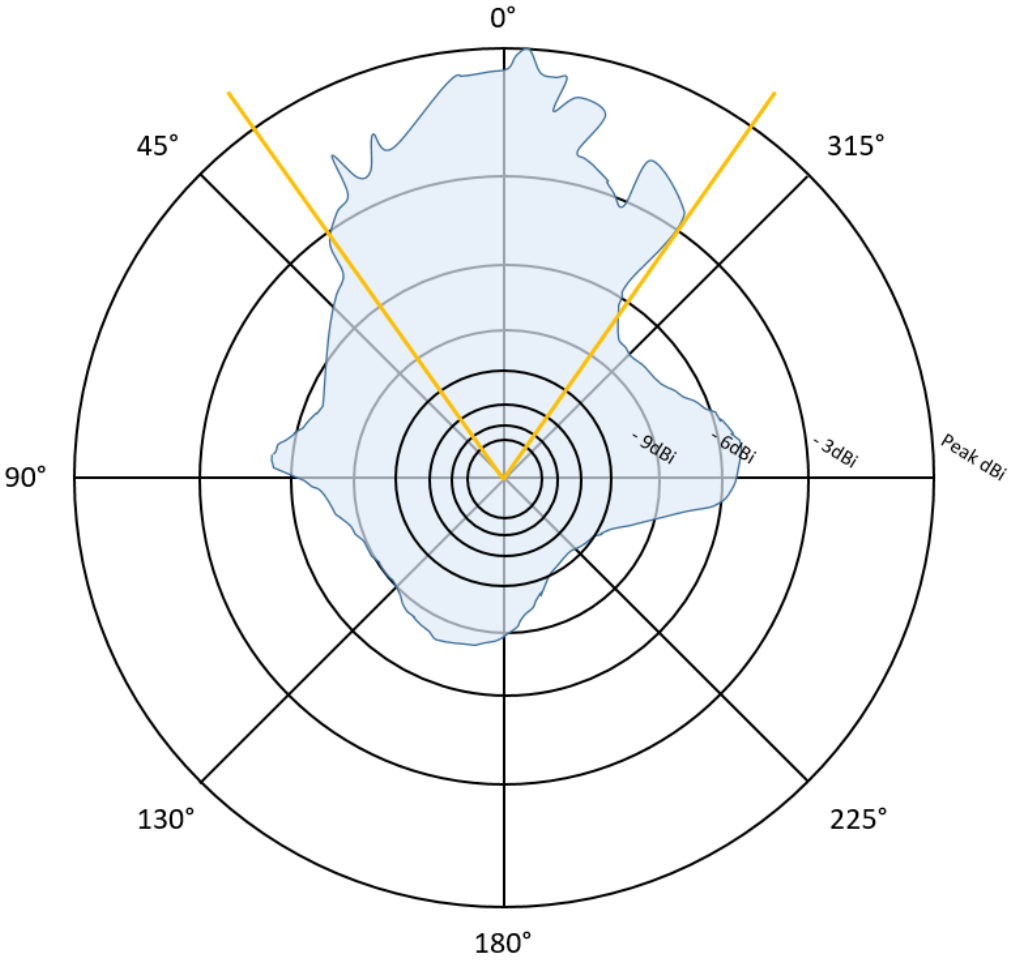



Fig. 4.1. A depiction of a possible antenna pattern with -3 dBi cuttoff points for main-lobe beam width.

peak side-lobe [32]. Due to the main-lobe's importance in direct communication and to reduce the complexity of the problem space, the main-lobes are only considered for this project.

## 4.3 Implementation

To implement this design, there are several modifications that must be made. First, the PSO must be modified to not only consider $x$, $y$, $z$, and frequency, but also the antennas RF reference heading. Second, an antenna model must be implemented and applied to the cost function. These implementations must be maintained with minimal effect on the performance of solution convergence time.

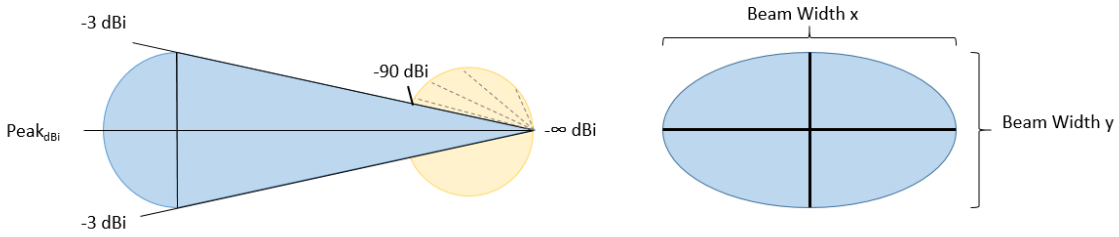



Fig. 4.2. Shown above is the concept for the directed antenna. Peak gain is calculated for the center of the beam width, and a parabolic roll-off is introduced until -3 dBi. A gradient for guiding the PSO to the main beam is introduced after the beam-width by using a linear gradient, shown in yellow, between -90 dBi and -192 dBi.

### 4.3.1 PSO Modifications

Modification of the PSO for determining heading is straightforward. In this sense, it is determined that spherical coordinates is the best way to represent the direction of the antenna. Therefore, the additional parameters to add to the solution space for each asset is $\theta$ for horizontal angle and $\phi$ with vertical angle. The terrain has a reference of direction for $x$, $y$, and $z$. Orientation is centered at (1,0,0). Rotations along $\theta$ and $\phi$ are associated with this reference. This results in the final solution space for each particle to be characterized by $(x, y, z, \theta, \phi)$.

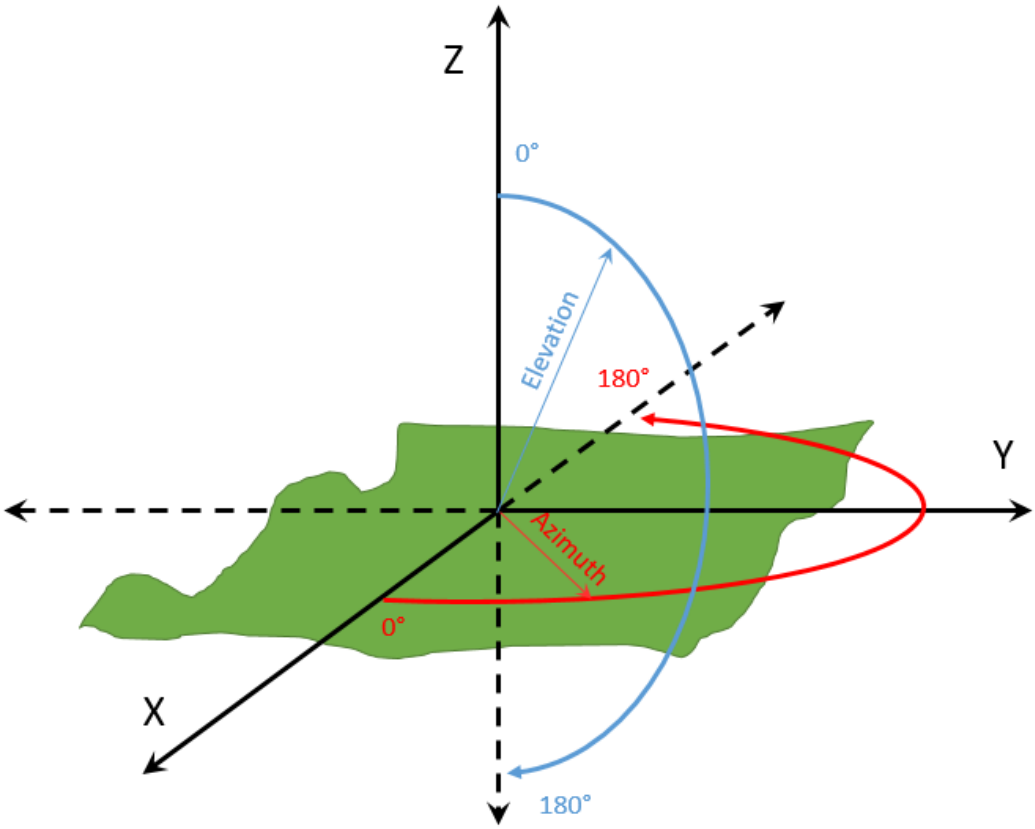



Fig. 4.3. Shown is the reference of orientation for azimuth and elevation.

To add $\theta$ and $\phi$ to the solution space, a simple modification needed to be made to the code. The PSO represents the values of x, y, z, and frequency as particles in a contiguous list, multiplied by the asset identification number. Figure 4.4 illustrates this representation. Each particle element is one of the six parameters associated with a specific asset number. Therefore, the dimension for each particle is shown in Equation 4.1. By changing the number of solution elements from four to six, initialization of the solution space is complete.

$$particle\ dimensions = (solutions\ elements) \times (assets) \tag{4.1}$$

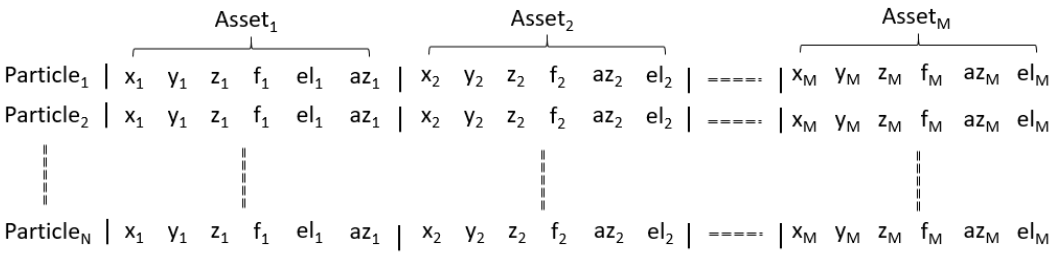



Fig. 4.4. Shown again is the particle representation of the PSO. Each particle contains the spatial, frequency, azimuth, and elevation information for each asset.

A list of parameters in each epoch and particle is passed to the *FitnessFunction-SpatialReciver::getCost()* function. It is at this point where the particles are translated into *XcvrObjects*, and antenna models are built with the new $\theta$ and $\phi$ rotations.

### 4.3.2 Antenna Models

With the implementation of *XcvrObject*, *PhysicalObject*, and *PhysicalOrientation* discussed in Chapter 2, antenna models are ready to be implemented. As stated previously, the current primary focus is to implement the main-lobe portion of the directional antenna without heavily affecting the performance and complexity of the PSO solution. Yet, future models may require for the implementation of more advanced concepts. Therefore, it is relevant to provide an abstract class to build more specific classes from, which only require limited interfacing.

An abstract class is considered, called *AbstractAntennaModel*. The implementation is shown in Figure 4.5. The virtual methods required for the developer to override is the function for calculating the gain based on an objects physical position. Because *AbstractAntennaModel* inherits a *PhysicsObject*, it also contains spatial positioning information and is able to return the gain of its propagation based on the relative position of other objects spatially. More detailed gain calculations are left for the implementation of derived classes.

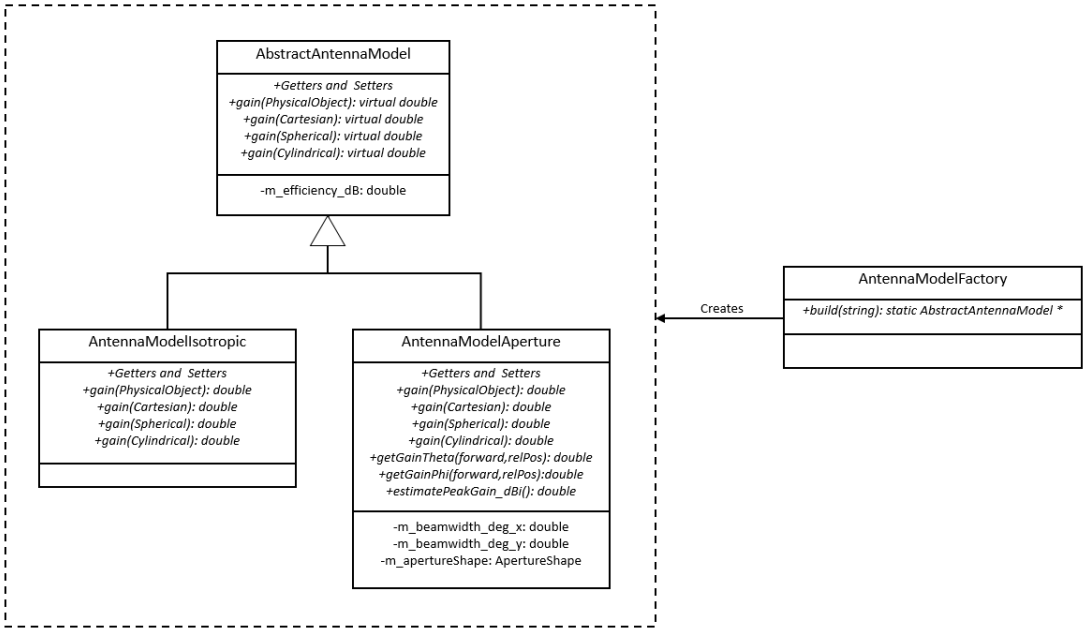



Fig. 4.5. Shown is the UML for the *AntennaModelFactor*, *AbstractAntennaModel*, and the derived *AntennaModelIsotropic* and *AntennaModelAperture*.

As a base case to maintain consistency with the original design of the project, an isotropic antenna is realized with the *AntennaModelIsotropic* class. The responsibility of this class is to return a value of 1.0 for the gain, regardless of which position is passed through the interface. Because the interface passes positions by reference, there should be no major performance impact from this modification. Therefore, the program can be tested against the original implementation with the use of isotropic antennas to measure any costs affected by introducing the orientation parameters to the solution space.

Implementation of the directional antennas occurs with the derived class *AntennaModelAperture*. The concept of directional antenna is shown in Figure 4.2. The model takes beam-width information for both horizontal and vertical portions of the feedhorn. These values are used to approximate the overall gain, based on a peak-gain value [32], estimated by Equation (4.2), and the beam-widths. The gain changes

throughout the beam-width is determined by a parabolic approximation, where angle differences of 0 in the x and y directions result in peak-gain, and angles at the edge of the beam-width result in -3 dBi gain. This parabolic approximation is shown in Equations 4.3 and 4.5. Equation 4.3 shows the calculation of individual coordinates with respect to x or y directions, and Equation 4.5 combines these two components and converts to dBi.

$$Gain_{Peak,dBi} = \frac{26000.0}{BW_x \times BW_y} \tag{4.2}$$

$$K(\theta_{rel}) = \begin{cases} 1 - 2(\frac{\theta_{rel}}{BW})^2 & |\theta_{rel}| \leq BW \\ (1 - |\frac{\theta_{rel}}{180.0}|) \times 10^{-9} & otherwise \end{cases} \tag{4.3}$$

$$|\theta_{rel}| \leq 180.0 \tag{4.4}$$

$$Gain_{dBi}(\theta_{rel,x}, \theta_{rel,y}) = Gain_{Peak,dBi} + 5\log(K^2(\theta_{rel,x}) \times K^2(\theta_{rel,y})) \tag{4.5}$$

Simply relying on the main-beam to guide the antenna is not enough, due to the introduction of discrete cutoffs to the fitness space. At the edge of the directed antenna's -3 dBi limit, a gradient is provided similar to pheromones in Chapter 3. From the edge of the beam-width to 180°, a linear gradient is introduced from -90 dBi to -192 dBi. This gradient is provided separately for both the x and y directions.

### 4.3.3 Culmination

Calculation of the cost is achieved by initializing each *XcvrObject* asset with a pointer to a receiver *AbstractAntennaModel* implementation, which is set by instantiating a new antenna model of either *AntennaModelIsotropic* or *AntennaModelAperture* with the current position and orientation found as a potential solution by the PSO. The function *checkOverPower()* calculates the free-space loss budget. The final gain of the received power is calculated by adding the gain from the antenna to the free-space loss budget equation. The full complexity of the antenna model is encap-

sulated in the derived antenna model class used for calculating gain, resulting in no major change to the original code-base.

Display of the solution points of the assets in 3D is now modified from black points to arrows facing the direction of orientation. This is shown in the 3D display of the assets, resulting in an intuitive and qualitative representation of the found solution, shown in Figure 4.6. The direction of the arrows indicate the beam path. Future work may encompass adding antenna beams to the 3D and 2D display.

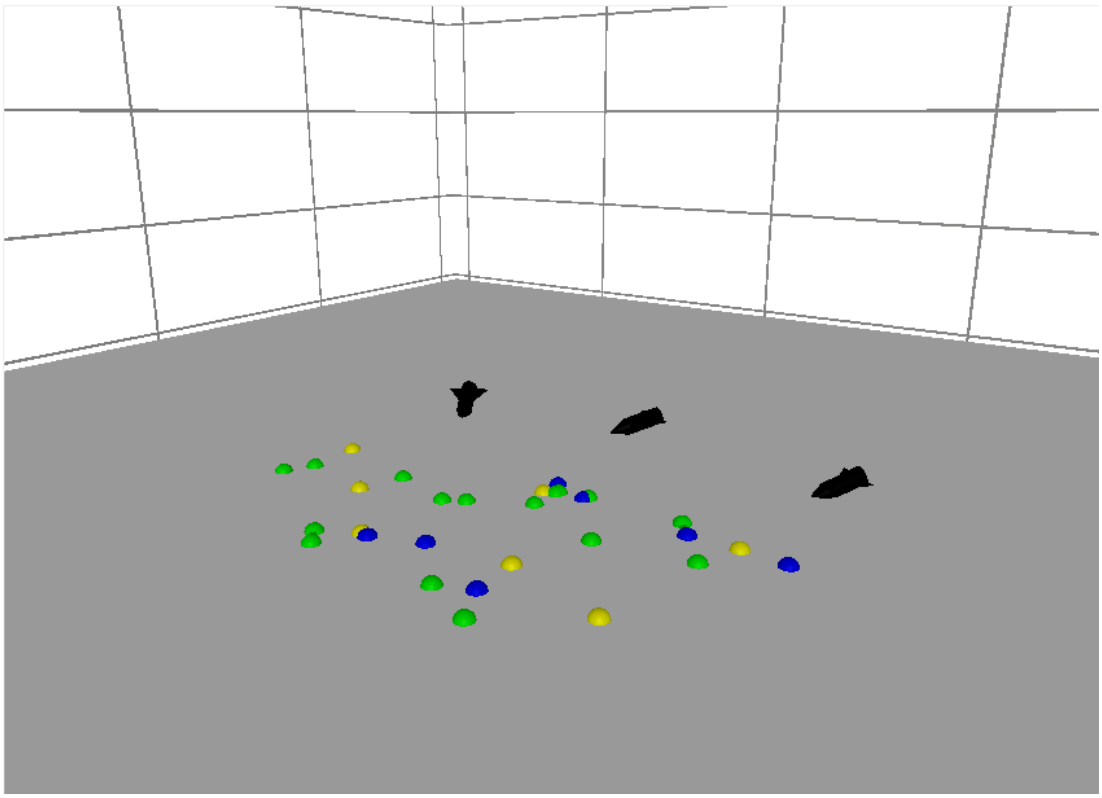


Fig. 4.6. An example of a solution using directed antennas.

# 5. META-PSO

## 5.1 Purpose

The fitness function is an objective measure of the utility of each potential solution to the problem space. In this work, fitness, cost, objective, and evaluation function all name the same function: a function determined to model the true utility function. A good fitness function has two properties: the fitness of each potential solution is highly correlated with its utility and the evaluation of the fitness function is computationally quick [33]. The problem with the EWAAP is that the utility is highly subjective; coverage of transmitters by the assets is driven by priority, power constraints, terrain constraints, and mission objectives. Although it is understood what a tremendously bad situation looks like, where no assets converge on an assignment, it is difficult to actually judge two potential solutions of similar assignments for which has the greater utility. It may be that in one situation, we require for the assets to be closer to a particular target, whereas in other situations we require for the asset to be out of sight.

One such utility that is of importance is in solution repeatability. In the case of one-off solutions, repeatability isn't an issue; we only need to know a good asset assignment to be implemented. In the case of real-time simulation, repeatability is essential; assets can only move so far in reality, resulting in that highly diverse solutions cannot be achieved. One such problematic area of repeatability can be handled by modifying the fitness function to take into account the physical constraints of the assets themselves. Another approach is to adjust the weights of the fitness function in order to produce results with lower variance. Optimization of the fitness function is examined through the use of Meta-PSO to not only increase repeatability, but to also reduce the convergence time cost.

The fitness function, shown again in 5.1 and explained in 1.4, can have direct consequences on the performance of the convergence of the solution, with respect to convergence speed, repeatability, accuracy, and whether or not it becomes trapped in a local maxim. Improper adjustments to the weights of each component may result in the solution becoming out of balance, with too much emphasis on one particular aspect over another. This may result in high variance across solutions or completely invalid solutions. Furthermore, due to the termination constraints on maximum number of epochs and a minimum delta, the overall runtime can directly be affected. As such, it was proposed that the fitness function be optimized in some fashion.

$$Fitness_{overall} = \sum_{C \in Components} \alpha_C fitness_C \tag{5.1}$$

## 5.2 Concept

Up to this point, the fitness function was manually adjusted by hand and observations were made about the variance and qualitative nature of each solution. It is necessary to provide a better, more quantitative, method to fine-tune the fitness function for the appropriate weights. Fine-tuning of parameters is a problem which optimization is specifically designed for. As such, PSO is considered for optimization of the fitness weights, by implementing a meta-fitness based on variance and overall runtime. Initial implementations of the Meta-PSO resulted in the requirements of actually considering normalization factors for fitness and some sense of the true objective of the original utility: signal coverage. Furthermore, violations were considered to reduce the breaking of certain parameters, such as terrain, altitude, pheromone, and keep-away constraints. Meta-PSO is shown to successfully choose weights for the main fitness function with lower variance and runtime, while maintaining valid solutions. The implementation of this work is presented here.

## 5.3 Implementation

Modifications were made to the program to implement the Meta PSO. A function for executing running of the PSO, *MainWindow::btn_runClicked()*, was already created, which setup the PSO, run the PSO until termination, and provide the overall runtime of the solution. Statistics, as defined in Chapter 2, were implemented to collect information about each particular run. These statistics were able to collect information about the variance across each asset's spatial position, which are of the most importance due to being the most difficult to change in short notice. Runtime statistics is also collected to provide the average runtime.

A function in *FitnessFunctionSpatialReceiver (FFSR)*, called *FFSR::validate()*, was created to validate the solution. This function validates the solution to ensure that pheromones are respected, the feedhorn is not overpowered, keep-away lines are not crossed, receiver-signal assignments are not blocked or underpowered, and all receivers are given at least one assignment.

A Monte Carlo simulation is used to gather statistics for each run. Targets are given a specific central position and range within that position to populate, but are populated randomly within that area. For a static number of iterations, the field is randomized in position and frequency and the PSO is solved. Each particle runs this number of iterations, and execution of the Meta PSO occurs for a given number of maximum epochs. Each particle is judged by the fitness function shown in Equation 5.2.

$$Fitness_{Meta} = (F_{variance})(F_{runtime}) + F_{power} + F_{coverage} + F_{violations} \tag{5.2}$$

$$F_{variance} = \frac{fitness_{mean}}{(\sum_{x \in D} var_x)(\sum_{w \in W} w + 1)} \tag{5.3}$$

$$D = \{x | x \in X, Y, Z\} \tag{5.4}$$

$$W = \{w | w \in Fitness\ Weights\} \tag{5.5}$$

$$F_{runtime} = \frac{1}{(runtime_{mean})^2} \tag{5.6}$$

$$F_{power} = \frac{sum_{RxPower}}{sum_{TxPower}} \tag{5.7}$$

$$F_{coverage} = \frac{1}{N} \sum_{i \in N} (Sigs_{assigned})_i \tag{5.8}$$

$$F_{violations} = (1 - \frac{violations}{violations_{potential}}) \tag{5.9}$$

$$\tag{5.10}$$

## 5.4 Usage

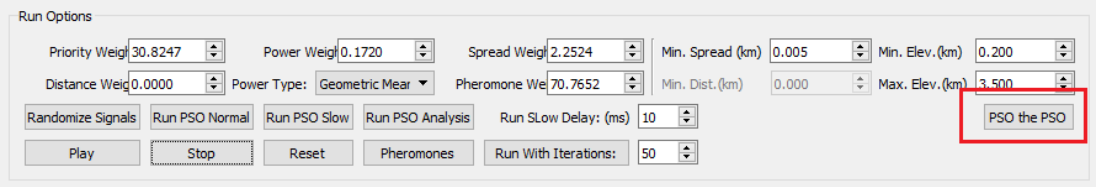



Fig. 5.1. Meta-PSO is implemented by clicking the "PSO the PSO" button in the Run Options on the Main GUI.

Execution of the Meta-PSO is began by clicking the Meta-PSO button, shown in Figure 5.1. A pop-up menu is presented, which gives the options shown in Figure 5.2. The options available are to change the total number of particles to use, the total iterations to execute for each evaluation, the total number of neighbors to use, the termination delta, and the overall number of iterations to fly each particle. The

termination is optional and can be enabled for a limit on total iterations or minimum delta. Monte Carlo can also be enabled or disabled, resulting in the training of the PSO on a single solution. A total amount of runs is presented, reminding the user of the consequence of executing Meta-PSO. An estimation of the runtime is presented, based on the current average runtime for each solution and the presented settings.

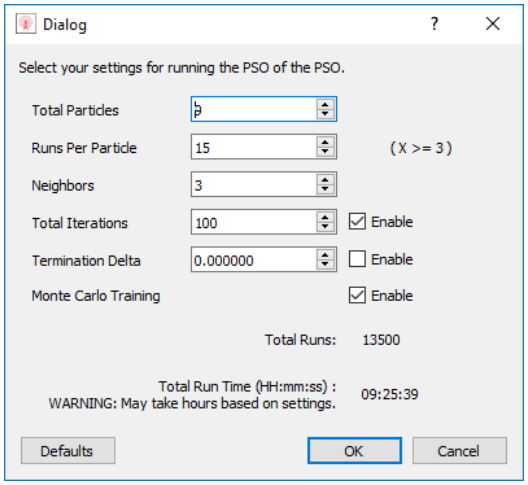



Fig. 5.2. Meta-PSO menu options.

After the Meta-PSO has completed, a file is written presenting the global best, as well as the personal best of all particles. Due to the time-cost of execution, a small amount of particles and a low number of iterations is typically used, resulting in a reduced search capability for a large problem space. It is necessary to get the personal best for each particle for further analysis after the termination of the Meta-PSO, to ensure that the best weights have not been missed.

# 6. RESULTS

A method must be implemented to determine the results of all the changes and their effects on the performance. Furthermore, Meta-PSO must be validated to prove that it is capable of providing better results than prior methods. An experiment is written as a function that cycles through 3 different scenarios and 17 fitness function weights. The scenarios tested are shown in Table 6.1. The weights are a combination of manual settings and results from various Meta-PSO runs, shown in Table 6.3. Each scenario and weight setting is run to test directional antenna and isotropic antenna for the best set of weights. In each experiment, a total of 100 trials are run and statistics are collected for the runtime and solutions. Metrics collected for the runtime statistics are average, standard deviation, and confidence interval up to 95%. Metrics collected for the solution are assets $x$, $y$, and $z$ standard deviation for each experiment. K-means is executed on the solution statistics to associate each asset with their expected solution. The statistics are used to determine the best set of weights with the lowest solution variance and the most sufficient mean runtime.

Table 6.1.
Scenarios used for test run. Each test run contained different settings for pheromone location, attribute, and radius, as well as terrain details. Location is in the format $(x, y, z)$. Location and radius are in km.

| Name | Scenario | Terrain | Pheromone Location | Pheromone Radius |
|---|---|---|---|---|
| S1 | Keep-Out | Default | (-15,0,0) | 35 |
| S2 | Keep-In | Default | (15,0,0) | 35 |
| S3 | Keep-Out | Mountain | (40,50,0) | 20 |

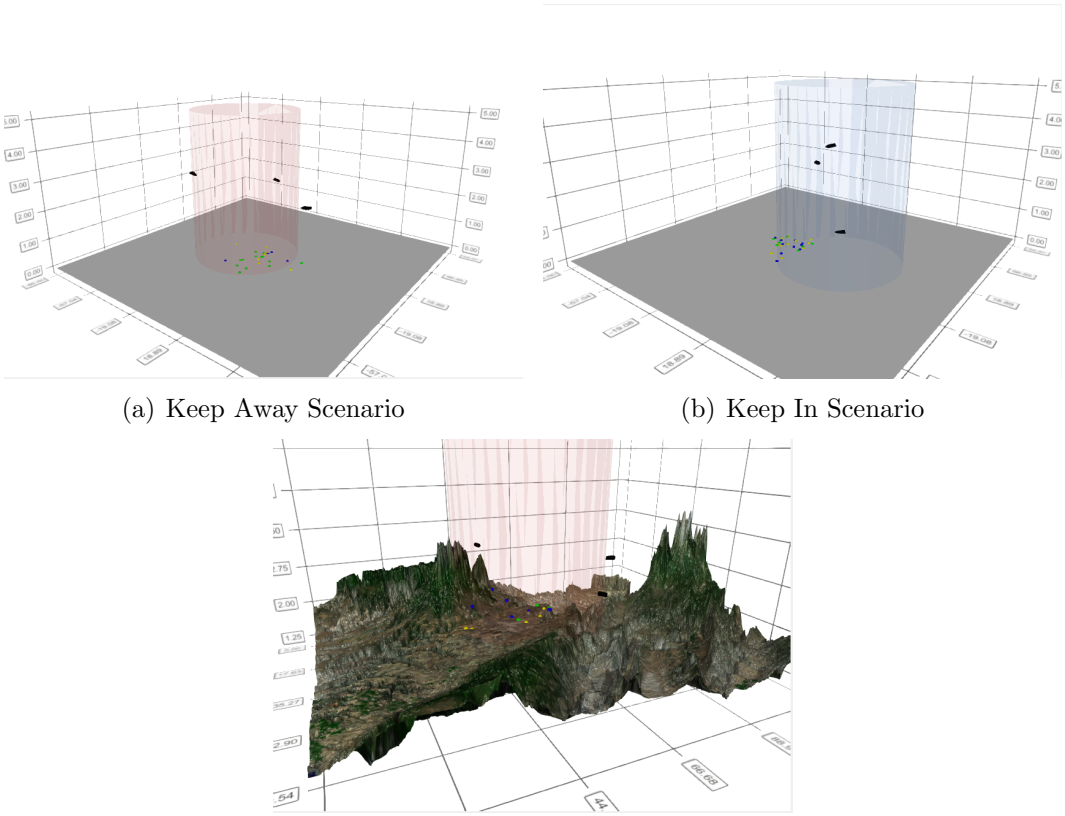
Table 6.2.
The parameters stayed the same across all scenarios, with the exception for antenna beam width, which is irrelevant for the isotropic antenna pattern scenarios.

| Parameter | Value |
|---|---|
| Signals | 30 |
| Receivers | 3 |
| Antenna Beam Width X | 180° |
| Antenna Beam Width Y | 15° |
| Tx Spread Radius | 30 |
| Frequency Step | 0.10 MHz |
| Receiver Bandwidth | 10 MHz |
| Receiver RF Frontend Limit | 5 dBm |
| Receiver RF Sensitivity | -88 dBm |
| Max Generations | 1000 |
| Swarm Termination Fitness Slope | 0.01 |
| Swarm Termination Window Size | 50 |
| Swarm Population Size | 200 |
| Swarm Neighbors | 20 |

Table 6.3.
Displayed are the weights gathered over a period of time. The description gives brief information that led to the development of new weights. The Training indicates whether the weights were tuned manually by hand, or by Meta-PSO. The Variance, Runtime, Scaling, SNR, and Violation columns indicates whether these were applied to the Meta-PSO fitness.

| Description | Training | Variance | Runtime | Scaling | SNR | Violations | Zone Type | Priority | Power | Spread | Distance | Pheromone |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pre-Pheromone | Manual | True | False | False | False | False | Legacy | 6 | 0.1 | 75 | 1 | 0 |
| Manual 1 | Manual | False | False | False | False | False | Pheromone | 6 | 0.1 | 75 | 1 | 0.1 |
| Manual 2 | Manual | False | False | False | False | False | Pheromone | 6 | 0.1 | 75 | 1 | 1 |
| Manual 3 | Manual | False | False | False | False | False | Pheromone | 6 | 0.1 | 75 | 1 | 10 |
| Manual 4 | Manual | False | False | False | False | False | Pheromone | 6 | 0.1 | 75 | 1 | 100 |
| Initial | Meta-PSO | True | False | False | False | False | Pheromone | 82.0594 | 2.0512 | 0.9755 | 12.9728 | 457.298 |
| Add Runtime | Meta-PSO | True | True | False | False | False | Pheromone | 94.9178 | 6.7238 | 51.6139 | 20.5778 | 779.1554 |
| Add Scaling and SNR | Meta-PSO | True | True | True | False | False | Pheromone | 11.5179 | 3.6895 | 26.1541 | 4.3484 | 75.0202 |
| Add Power Ratio | Meta-PSO | True | True | True | True | False | Pheromone | 99.2189 | 44.1664 | 0.7956 | 0 | 35.5728 |
| Subtract Violations Geo | Meta-PSO | True | True | True | True | True | Pheromone | 74.1668 | 42.836 | 12.563 | 0 | 270.3725 |
| Subtract Violations Sum | Meta-PSO | True | True | True | True | True | Pheromone | 51.6092 | 29.9093 | 48.8893 | 0 | 66.3717 |
| Multiply Violations Geo | Meta-PSO | True | True | True | True | True | Pheromone | 90.5799 | 108.0496 | 55.2396 | 0 | 1.9505 |
| Add (1-Violations) Geo | Meta-PSO | True | True | True | True | True | Pheromone | 55.3883 | 70.9463 | 13.717 | 0 | 174.239 |
| Add (1-Violations) Sum | Meta-PSO | True | True | True | True | True | Pheromone | 83.6944 | 7.5139 | 6.2293 | 0 | 98.5104 |
| Modify Antenna 0 | Meta-PSO | True | True | True | True | True | Pheromone | 59.5821 | 1.0869 | 8.5821 | 0 | 74.5062 |
| Modify Antenna 1 | Meta-PSO | True | True | True | True | True | Pheromone | 87.6669 | 6.56875 | 9.37458 | 0 | 99.0534 |
| Modify Antenna 2 | Meta-PSO | True | True | True | True | True | Pheromone | 30.8247 | 0.172028 | 2.25244 | 0 | 70.7652 |

Each scenario is run using the same settings as shown in Table 6.2. Figure 6.1 shows the scenarios used for each experiment. The first scenario in Figure 6.1(a) is a keep-away pheromone Zone with an attracting Beacon centralized on a set of 30 transmitters. The keep-away line and circle are disabled, except for the test case where pheromone strength is 0. The test case with 0 pheromone strength are the original weights found by [2]. The keep-away pheromone is set for a radius of 35 km and centered at (-15, 0, 0). No terrain is used for this example.



(a) Keep Away Scenario

(b) Keep In Scenario

(c) Mountain Scenario

Fig. 6.1. Displayed are the scenarios used for gathering solution asset variance and mean runtime.

The second scenario shown in Figure 6.1(b) shows the case for a keep-in pheromone Zone and an attracting Beacon. The keep-in zone uses the same 35 km radius, but uses the keep-away line for the base case. For this scenario, a terrain is also not used.

The third and final scenario shown in Figure 6.1(c) shows the case for a mountainous terrain with a keep-away zone. In this case, line-of-sight becomes an issue for the PSO to overcome and a more dynamic solution space is presented.

Table 6.4.
Shown below are the results for running the experiments on the isotropic antennas.

| # | Priority | Power | Spread | Distance | Pheromone | Position Std. Dev. | Avg. Runtime (ms) | Runtime Std. Dev. | Runtime CI (95%) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 0.1 | 75 | 1 | 0 | 20.2083 | 444.1250 | 116.8849 | 23.1899 |
| 2 | 6 | 0.1 | 75 | 1 | 0.1 | 17.1896 | 466.6000 | 127.5830 | 25.3125 |
| 3 | 6 | 0.1 | 75 | 1 | 1 | 17.7423 | 445.1600 | 116.2306 | 23.0602 |
| 4 | 6 | 0.1 | 75 | 1 | 10 | 17.8306 | 440.1250 | 120.1508 | 23.8379 |
| 5 | 6 | 0.1 | 75 | 1 | 100 | 16.5363 | 465.5850 | 117.7706 | 23.3657 |
| 6 | 82.0594 | 2.0512 | 0.9755 | 12.9728 | 457.298 | **12.7007** | 765.4350 | 173.1827 | 34.3594 |
| 7 | 94.9178 | 6.7238 | 51.6139 | 20.5778 | 779.1554 | 13.0737 | 813.2150 | 206.4757 | 40.9648 |
| 8 | 11.5179 | 3.6895 | 26.1541 | 4.3484 | 75.0202 | 13.2972 | 573.1150 | 144.3131 | 28.6317 |
| 9 | 99.2189 | 44.1664 | 0.7956 | 0 | 35.5728 | 15.0005 | 388.3550 | 86.6896 | 17.1992 |
| 10 | 74.1668 | 42.836 | 12.563 | 0 | 270.3725 | 15.0059 | 453.9850 | 120.1227 | 23.8324 |
| 11 | 51.6092 | 29.9093 | 48.8893 | 0 | 66.3717 | 28.3099 | 465.7950 | 136.8658 | 27.1542 |
| 12 | 90.5799 | 108.0496 | 55.2396 | 0 | 1.9505 | 24.8178 | 508.7450 | 149.0206 | 29.5657 |
| 13 | 55.3883 | 70.9463 | 13.717 | 0 | 174.239 | 16.2862 | 460.8200 | 122.7775 | 24.3591 |
| 14 | 83.6944 | 7.5139 | 6.2293 | 0 | 98.5104 | 16.1227 | 429.9900 | 98.2441 | 19.4916 |
| 15 | 59.5821 | 1.0869 | 8.5821 | 0 | 74.5062 | 17.8153 | 422.6950 | 98.9251 | 19.6268 |
| 16 | 87.6669 | 6.56875 | 9.37458 | 0 | 99.0534 | 16.3915 | 424.6650 | 111.5468 | 22.1309 |
| 17 | 30.8247 | 0.172028 | 2.25244 | 0 | 70.7652 | 15.2577 | **375.5250** | 78.9214 | 15.6580 |

The results for isotropic and aperture antennas are shown separately in Tables 6.4 and 6.5 respectively. For the isotropic antenna, the most accurate weights that meet the under 1 second benchmark are set #6 at 12.7007 standard deviation, found by the first implementation of Meta-PSO that only optimized for variance. This would make sense due to the minimization of variance being the most important aspect of this version of Meta-PSo. Also shown in the isotropic antenna is that despite variance being low, runtime had tremendously increased and came close to violating the 1 second benchmark at a mean of 765.4350±34.3594 ms. The fastest set of weights are the most recent set #17 at 375.5250±15.6580 ms, which optimize for variance, runtime, received signal power, and minimized violations. Not only is this set the fastest, but the standard deviation of the solutions points are close to the median of the results at 15.2577.

The result for aperture antenna are shown in Table 6.5. Now, it is shown that despite having the lowest variation, set #6 violates the 1 second benchmark. The

Table 6.5.
Shown below are the results for running the experiments on the aperture antennas.

| # | Priority | Power | Spread | Distance | Pheromone | Position Std. Dev. | Avg. Runtime (ms) | Runtime Std. Dev. | Runtime CI (95%) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 0.1 | 75 | 1 | 0 | 37.1158 | 686.1917 | 275.9617 | ±54.7508 |
| 2 | 6 | 0.1 | 75 | 1 | 0.1 | 36.9525 | 756.3883 | 300.5695 | ±59.6329 |
| 3 | 6 | 0.1 | 75 | 1 | 1 | 31.0767 | 723.6217 | 264.6405 | ±52.5047 |
| 4 | 6 | 0.1 | 75 | 1 | 10 | 30.0297 | 736.8867 | 260.0175 | ±51.5875 |
| 5 | 6 | 0.1 | 75 | 1 | 100 | 28.8552 | 747.7517 | 264.7351 | ±52.5234 |
| 6 | 82.0594 | 2.0512 | 0.9755 | 12.9728 | 457.298 | **14.3593** | **1242.1750** | 957.8861 | ±190.0445 |
| 7 | 94.9178 | 6.7238 | 51.6139 | 20.5778 | 779.1554 | **14.6189** | **1280.2000** | 431.7883 | ±85.6668 |
| 8 | 11.5179 | 3.6895 | 26.1541 | 4.3484 | 75.0202 | 24.0224 | **997.9750** | 357.4511 | **±70.9183** |
| 9 | 99.2189 | 44.1664 | 0.7956 | 0 | 35.5728 | 16.3570 | 620.2150 | 155.8476 | ±30.9202 |
| 10 | 74.1668 | 42.836 | 12.563 | 0 | 270.3725 | 15.5961 | 783.2583 | 186.2255 | ±36.9471 |
| 11 | 51.6092 | 29.9093 | 48.8893 | 0 | 66.3717 | 33.8966 | 684.8633 | 205.1911 | ±40.7099 |
| 12 | 90.5799 | 108.0496 | 55.2396 | 0 | 1.9505 | 34.9286 | 734.0400 | 225.5896 | ±44.7570 |
| 13 | 55.3883 | 70.9463 | 13.717 | 0 | 174.239 | 17.7004 | 727.7300 | 186.9420 | ±37.0893 |
| 14 | 83.6944 | 7.5139 | 6.2293 | 0 | 98.5104 | 16.7673 | 659.4017 | 158.8739 | ±31.5206 |
| 15 | 59.5821 | 1.0869 | 8.5821 | 0 | 74.5062 | 19.2514 | 606.3983 | 153.9190 | ±30.5375 |
| 16 | 87.6669 | 6.56875 | 9.37458 | 0 | 99.0534 | 18.1400 | 668.5667 | 170.5716 | ±33.8414 |
| 17 | 30.8247 | 0.172028 | 2.25244 | 0 | 70.7652 | **15.5213** | **601.9083** | 144.0901 | ±28.5875 |

next best set which doesn't violate the runtime benchmark is also the fastest set, and once again set #17, with a solution standard deviation of 15.5213 and an average runtime of 601.9083±28.5875 ms.

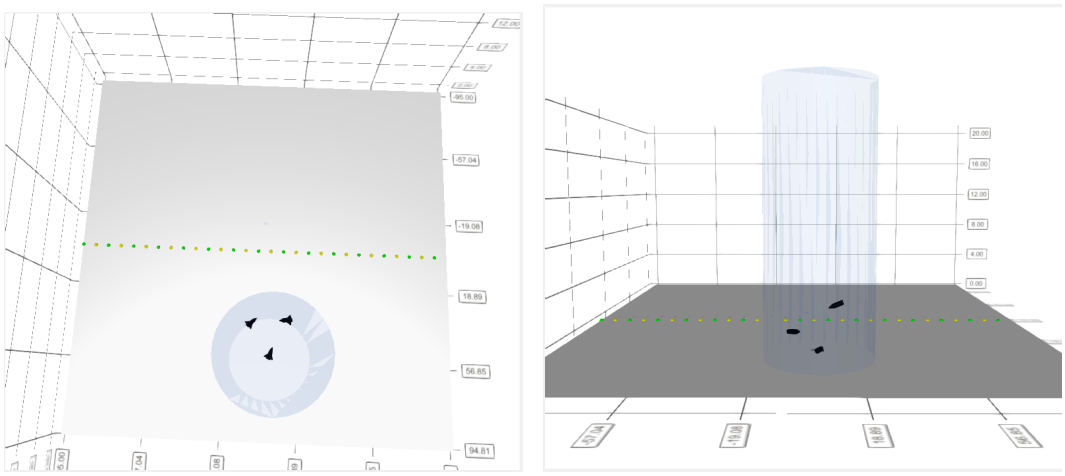
Table 6.6.
Shown below are the total combined results for both aperture and isotropic antennas.

| # | Priority | Power | Spread | Distance | Pheromone | Position Std. Dev. | Avg. Runtime (ms) | Runtime Std. Dev. | Runtime CI (95%) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 0.1 | 75 | 1 | 0 | 29.8828 | 565.1583 | 116.8849 | 23.1899 |
| 2 | 6 | 0.1 | 75 | 1 | 0.1 | 28.8182 | 611.4942 | 127.5830 | 25.3125 |
| 3 | 6 | 0.1 | 75 | 1 | 1 | 25.3036 | 584.3908 | 116.2306 | 23.0602 |
| 4 | 6 | 0.1 | 75 | 1 | 10 | 24.6952 | 588.5058 | 120.1508 | 23.8379 |
| 5 | 6 | 0.1 | 75 | 1 | 100 | 23.5167 | 606.6683 | 117.7706 | 23.3657 |
| 6 | 82.0594 | 2.0512 | 0.9755 | 12.9728 | 457.298 | **13.5554** | **1003.8050** | 173.1827 | 34.3594 |
| 7 | 94.9178 | 6.7238 | 51.6139 | 20.5778 | 779.1554 | **13.8678** | **1046.7075** | 206.4757 | 40.9648 |
| 8 | 11.5179 | 3.6895 | 26.1541 | 4.3484 | 75.0202 | 19.4151 | 785.5450 | 144.3131 | 28.6317 |
| 9 | 99.2189 | 44.1664 | 0.7956 | 0 | 35.5728 | 15.6934 | 504.2850 | 86.6896 | 17.1992 |
| 10 | 74.1668 | 42.836 | 12.563 | 0 | 270.3725 | 15.3038 | 618.6217 | 120.1227 | 23.8324 |
| 11 | 51.6092 | 29.9093 | 48.8893 | 0 | 66.3717 | 31.2284 | 575.3292 | 136.8658 | 27.1542 |
| 12 | 90.5799 | 108.0496 | 55.2396 | 0 | 1.9505 | 30.2979 | 621.3925 | 149.0206 | 29.5657 |
| 13 | 55.3883 | 70.9463 | 13.717 | 0 | 174.239 | 17.0080 | 594.2750 | 122.7775 | 24.3591 |
| 14 | 83.6944 | 7.5139 | 6.2293 | 0 | 98.5104 | 16.4482 | 544.6958 | 98.2441 | 19.4916 |
| 15 | 59.5821 | 1.0869 | 8.5821 | 0 | 74.5062 | 18.5472 | 514.5467 | 98.9251 | 19.6268 |
| 16 | 87.6669 | 6.56875 | 9.37458 | 0 | 99.0534 | 17.2878 | 546.6158 | 111.5468 | 22.1309 |
| 17 | 30.8247 | 0.172028 | 2.25244 | 0 | 70.7652 | 15.3901 | **488.7167** | 78.9214 | 15.6580 |

Table 6.6 shows the combined results for both isotropic and aperture antenna results. It is shown that the best performing set of weights is set #17, with a solution standard deviation 15.3901 and an average runtime of 488.7167±15.6580 ms.

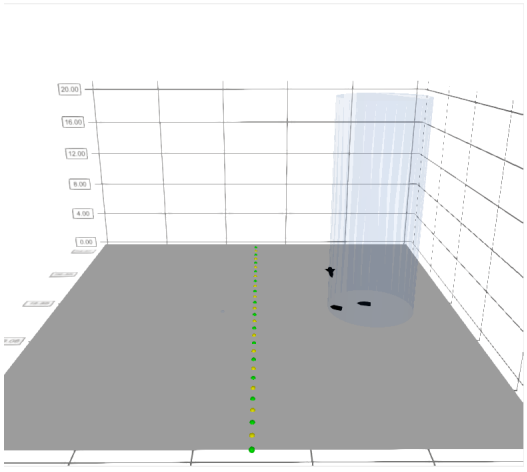
A final verification can be made by running the new weights against the ground-truth. In the ground-truth, transmitters are lined up and spaced equally apart in spatiality and frequency. The spacing is equidistant across the map along the $y$-axis, and one MHz for the frequency domain. Priorities are alternated between 3 and 5. In the ground-truth, all the signals must be assigned to the receivers without overlap. Figures 6.2 and 6.3 shows the result of the ground-truth against the new weights.


Fig. 6.2. Shown are the results of the new weight ran against the ground-truth.



(a) 3D Top

(b) 3D Rear



(c) 3D Side

Fig. 6.3. Shown are the 3D results of the ground truth.

## 6.1 Discussion

The results have shown that the most recent modifications to the Meta-PSO, as presented in Chapter 5, produce the best set of weights with respect to runtime and solution precision. The Meta-PSO outperforms the hand-chosen set of weights provided by prior work. This shows that Meta-PSO is an effective approach to optimizing the fitness function weights for the internal PSO. One thing to note is that runtime is typically much longer than prior work. Crespo [2] presented an average runtime of

178 ms, which is close to half that of 375.5250 ms. There are several reasons for this increase, all due to the added functionality provided in this research. The longest computation is the Line of Sight (LOS) check which increases time-complexity by adding an extra dimension of checking terrain for blocking. Other lesser computations are added for statistics calls for getCost() and 3D visualization. Despite the increase in runtime cost, increased realism for the project is achieved. Due to the optimization of the Meta-PSO, the performance under 1 second is maintained. From the Meta-PSO analysis, the distance fitness component now can be replaced by a Beacon Pheromone. And finally, the Meta-PSO is able to reduce the variance over the hand-picked weights while maintaining accuracy in the ground-truth tests.

# 7. SUMMARY

This work has successfully presented the usefulness and effectiveness of the modifications applied to the Electronic Warfare (EW) Asset Allocation Problem (EWAAP). Object-oriented programming (OOP) concepts has been implemented to provide more readable, reusable, and modular code. Human-swarm interactions has been implemented using the concepts of pheromone Zones and Beacons to allow a user to provide keep-in and keep-out areas to the problem space. Directional antenna has been implemented to provide more realistic and flexible RF propagation models. Optimization of the fitness function weights with Meta-PSO has been shown to provide faster and more precise solutions, resulting in a better performing PSO model. Solutions are able to converge on average in multiple scenarios in $488.7167 \pm 15.6580$ ms within a standard deviation of 15.3901 for asset positions.

# 8. RECOMMENDATIONS

It is recommended that work be continued in implementing more complex RF propagation and antenna models. Work should also be implemented in defining the assets as land, sea, and undersea assets, providing limitations on asset physics and spatial movements. It is recommended that these implementations be made by inheriting from *PhysicalObject* or *XcvrObject* to extend the capability of assets, and providing physics limitations via soft penalties to the fitness function.

The program should be modified to allow the user to select whether physics limitations are important for calculation, for real-time simulations, or if only the final solution is important. In this sense, it should be determined if a single, good solution is the most important aspect of the program, or if a highly repeatable solution for repeated realtime calculations of automated moving targets. If repeatability is of higher priority, the fitness function should take into consideration the physics limitations of the asset that it is calculating for, such as directional changes and acceleration limits.

Receiver-transmitter signal assignments should be re-evaluated to determine if a more dynamic signal assignment should be implemented. Currently, the closest signal to the receiver is assigned, but it may be possible that another receiver which is not heavily loaded would be capable of taking the assignment. If a better signal assignment scheme can be implemented, this may result in higher repeatability across solutions.

The implementation of the 3D surface using Qt Data Visualization is reaching its limit with the current scheme. Better 3D representations should be investigated, i.e. Unity or Unreal game/simulation engines. Qt Data Visualization is more geared to presenting static 3D models, rather than animations. There are currently problems in the code where transparent objects are not properly applying the correct Z-depth due to the underlying Qt render engine. In order to appropriately implement transparent

options, either the render engine will need to be reimplemented and modified, or a costly volumetric renderer will need to be implemented instead of the current custom object model.

More work can be implemented with providing more geometric shapes, or even custom shapes, for the pheromone Zones. Cylinder is implemented and Sphere would be another more simple modification, but more dynamic types of shapes will require extra consideration for collision detection and 3D presentation.

Finally, 2D and 3D beams should be provided to show the antenna patterns and the directions they are pointing.

# REFERENCES

# REFERENCES


[1] J. Reynolds, "Particle swarm optimization applied to real-time asset allocation," Purdue University MS Thesis, Indianapolis, IN, 2015.

[2] J. Crespo, "Asset allocation in frequency and in 3 spatial dimensions for electronic warfare application," Purdue University MS Thesis, Indianapolis, IN, 2016.

[3] L. Christopher, W. Boler, C. Wieczorek, J. Crespo, P. Witcher, S. A. Hawkins, and J. Stewart, "Asset allocation with swarm/human blended intelligence," in *2016 Swarm/Human Blended Intelligence Workshop (SHBI)*, October 2016, pp. 1–5.

[4] P. R. Witcher, "Particle swarm optimization in the dynamic electronic warfare battlefield," Purdue University MS Thesis, Indianapolis, IN, 2017.

[5] C. Wieczorek, "3D terrain visualization and cpu parallelization of particle swarm optimization (unpublished master's thesis)," Purdue University, Indianapolis, IN, 2018.

[6] R. C. Eberhart, J. Kennedy *et al.*, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1. New York, NY, 1995, pp. 39–43.

[7] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[8] R. C. Eberhart and J. Kennedy, "The particle swarm: social adaptation in information-processing systems," in *New ideas in optimization*. McGraw-Hill Ltd., UK, 1999, pp. 379–388.

[9] C. Wen and R. C. Eberhart, "Genetic algorithm for logistics scheduling problem," in *WCCI*. IEEE, 2002, pp. 512–516.

[10] Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, "Particle swarm optimization: basic concepts, variants and applications in power systems," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp. 171–195, 2008.

[11] J. Robinson and Y. Rahmat-Samii, "Particle swarm optimization in electromagnetics," *IEEE Transactions on Antennas and Propagation*, vol. 52, no. 2, pp. 397–407, 2004.

[12] A. Konak, G. E. Buchert, and J. Juro, "A flocking-based approach to maintain connectivity in mobile wireless ad hoc networks," *Applied Soft Computing*, vol. 13, no. 2, pp. 1284–1291, 2013.

[13] A. M. Naghsh, J. Gancet, A. Tanoto, and C. Roast, "Analysis and design of human-robot swarm interaction in firefighting," in *The 17th IEEE International Symposium on Robot and Human Interactive Communication, 2008. RO-MAN 2008.* IEEE, 2008, pp. 255–260.

[14] F. Johansson and G. Falkman, "Real-time allocation of defensive resources to rockets, artillery, and mortars," in *2010 13th Conference on Information Fusion (FUSION).* IEEE, 2010, pp. 1–8.

[15] I. Montalvo, J. Izquierdo, S. Schwarze, and R. Prez-Garca, "Multi-objective particle swarm optimization applied to water distribution systems design: An approach with human interaction," *Mathematical and Computer Modelling*, vol. 52, no. 7, pp. 1219 – 1227, 2010, last date accessed: 3/18/2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0895717710000695

[16] M. Iqbal and C. G. Freitas, Alex A.and Johnson, "Protein interaction inference using particle swarm optimization algorithm," in *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, E. Marchiori and J. H. Moore, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 61–70.

[17] A. E. Forooshani, A. A. Lotfi-Neyestanak, and D. G. Michelson, "Optimization of antenna placement in distributed mimo systems for underground mines," *IEEE Transactions on Wireless Communications*, vol. 13, no. 9, pp. 4685–4692, September 2014.

[18] J. Kennedy, "The particle swarm paradigm is a particle swarm," in *2016 Swarm/Human Blended Intelligence Workshop (SHBI)*, October 2016, pp. 1–5.

[19] A. Kolling, K. Sycara, S. Nunnally, and M. Lewis, "Human swarm interaction: An experimental study of two types of interaction with foraging swarms," *Journal of Human-Robot Interaction*, vol. 1, pp. 78–95, June 2013.

[20] D. Palmer, M. Kirschenbaum, E. Mustee, and J. Dengler, "Human-swarm hybrids outperform both humans and swarms solving digital jigsaw puzzles," in *2014 IEEE Symposium on Swarm Intelligence (SIS).* IEEE, 2014, pp. 1–8.

[21] L. Rosenberg, D. Baltaxe, and N. Pescetelli, "Crowds vs swarms, a comparison of intelligence," in *2016 Swarm/Human Blended Intelligence Workshop (SHBI)*, October 2016, pp. 1–4.

[22] A. D. I. Kramer, J. E. Guillory, and J. T. Hancock, "Experimental evidence of massive-scale emotional contagion through social networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 24, pp. 8788–8790, 2014, last date accessed: 3/18/2018. [Online]. Available: http://www.pnas.org/content/111/24/8788

[23] C. Veenhuis, "Advanced meta-pso," in *2006 Sixth International Conference on Hybrid Intelligent Systems (HIS'06)*, December 2006, pp. 54–54.

[24] M. Meissner, M. Schmuker, and G. Schneider, "Optimized particle swarm optimization (opso) and its application to artificial neural network training," *BMC Bioinformatics*, vol. 7, p. 125, 2006.

[25] S. Sakamoto, T. Oda, M. Ikeda, L. Barolli, F. Xhafa, and I. Woungang, "Investigation of fitness function weight-coefficients for optimization in wmn-pso simulation system," in *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, July 2016, pp. 224–229.

[26] Z. Ma and G. A. E. Vandenbosch, "Comparison of weighted sum approaches for pso fitness functions in antenna design," in *2012 9th European Radar Conference*, October 2012, pp. 516–519.

[27] T. Hendtlass, "Fitness estimation and the particle swarm optimisation algorithm," in *2007 IEEE Congress on Evolutionary Computation*, September 2007, pp. 4266–4272.

[28] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 2005.

[29] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.

[30] L. R. Da. Object oriented programming. Last date accessed: 3/18/2018. [Online]. Available: http://www.ctp.bilkent.edu.tr/ř̃russell/java/LectureNotes/1_OOConcepts.htm

[31] H. F. Mathis, "A short proof that an isotropic antenna is impossible," *Proceedings of the IRE*, vol. 39, no. 8, August 1951.

[32] J. S. Seybold, *Introduction to RF Propagation*. Wiley-Interscience, September 2005.

[33] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

VITA

# VITA

William Boler is a Computer Engineering student at Indiana University-Purdue University, Indianapolis (IUPUI). He is currently a graduate student with intentions to obtain a Master's in Computer Engineering, and is studying computational intelligence under the guidance of Dr. Lauren Christopher. William served in the United States Navy active-duty as an Information Systems Technician and learned about RF propagation through technical experience and training.